Hes·so // VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Domaine Sciences de l'ingénieur
Rte du Rawyl 47
CH- 1950 Sion 2
Tél. +41 27 606 85 11
Fax +41 27 606 85 75
info@hevs.ch

www.hevs.ch

Filière Systèmes industriels

Orientation Infotronics

Diplôme 2012

Alexandre Ganchinho

Hardware Sound processing unit (HSPU)

Professeur	Pierre-André Mudry
Expert	David Jilli

Sion, le 9 juillet 2012

HES-SO Valais

SI	TV
X	X

Données du travail de diplôme
Daten der Diplomarbeit

FO 1.2.02.07.AB
pof/31/01/2009

☒ FSI ☐ FTV	Année académique / *Studienjahr* **2011/2012**	No TD / *Nr. DA* **it/2012/19**
Mandant / *Auftraggeber* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire *Partnerinstitution*	Etudiant / *Student* **Alexandre Ganchinho** Professeur / *Dozent* **Pierre-André Mudry**	Lieu d'exécution / *Ausführungsort* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire *Partnerinstitution*
Travail confidentiel / *vertrauliche Arbeit* ☐ oui / ja [1] ☒ non / nein	Expert / *Experte (données complètes)* **David Jilli** Avenue Montagibert 8 | 1005 Lausanne	

Titre / *Titel*

Hardware Sound processing unit (HSPU)

Description et Objectifs / *Beschreibung und Ziele*

De nos jours, nombreux sont les appareils électroniques ayant besoin de produire du son. Certains d'entre eux, tels que les baladeurs et les téléphones portables ont besoin d'être spécialisés dans le décodage de flux audio compressés. D'autres, comme les claviers/racks synthétiseurs, les consoles de jeux et les ordinateurs doivent être capables de jouer une grande quantité de sons simultanés avec divers traitements.

Le but de ce projet consiste à concevoir un sound processing unit (SPU) en hardware se comportant comme un synthétiseur basé sur de la lecture d'échantillons audio non compressés (LPCM). Avec un tel système, l'utilisateur peut charger des échantillons audio en mémoire (correspondant à des instruments ou à des bruitages) et commander leur lecture à différentes fréquences, tout en y appliquant des modifications telles que le volume et la clarté (filtre passe-bas).

Ces modificateurs peuvent être contrôlés via des valeurs directes et via des modulateurs d'enveloppe (ADSR, LFO). Pour l'implémentation, le projet fait essentiellement appel à des notions de VHDL (pour la réalisation du SPU) et de programmation (C/C++, pour la réalisation de drivers).

Objectifs à atteindre :

— Architecture hardware pour un synthétiseur pipeliné réalisé en co-design
— Réalisation et tests du synthétiseur en VHDL
— Interfaçage du synthétiseur via MIDI
— Réalisation d'un démonstrateur permettant de jouer du son à partir d'une interface MIDI.

Délais / *Termine*

Attribution du thème / *Ausgabe des Auftrags:* 14.05.2012	Exposition publique / *Ausstellung Diplomarbeiten:* 31.08.2012
Remise du rapport / *Abgabe des Schlussberichts:* 09.07.2012 | 12h00	Défense orale / *Mündliche Verteidigung:* Semaine / *Woche 36*

Signature ou visa / *Unterschrift oder Visum*

Responsable de l'orientation
Leiter der Vertiefungsrichtung: .....

[1] Etudiant/*Student*: .....

[1] *Par sa signature, l'étudiant-e s'engage à respecter strictement la directive et le caractère confidentiel du travail de diplôme qui lui est confié et des informations mises à sa disposition.*
Durch seine Unterschrift verpflichtet sich der Student, die Richtlinie einzuhalten sowie die Vertraulichkeit der Diplomarbeit und der dafür zur Verfügung gestellten Informationen zu wahren.

Rapport reçu le / *Schlussbericht erhalten am* ..... Visa du secrétariat / *Visum des Sekretariats* .....

Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

HSPU - Hardware Sound Processing Unit

Diplômant/e Alexandre Ganchinho

Objectif du projet

Ce projet consiste à réaliser une Sound Processing Unit (SPU) en hardware ayant pour but d'accélérer les opérations nécessaires à la production de son et/ou de musique basés sur de la lecture massivement parallèle d'échantillons sonores.

Méthodes | Expériences | Résultats

La chaîne d'opérations nécessaire pour la production de son est très simple : prédiction des nouvelles positions, collecte des échantillons, interpolation, altération (amplitude, filtre, etc.), accumulation. Néanmoins, la fréquence de ces opérations est nettement trop élevée pour qu'elles soient réalisées avec efficience par un CPU embarqué. Les contraintes temporelles strictes qu'impose le rendu simultané de nombreux sons ont mené à la réalisation sur FPGA d'un SPU basé sur une architecture pipelinée. Ce SPU, disposant de sa propre mémoire RAM pour le stockage des échantillons, est directement contrôlable par le CPU sur lequel il est mappé.

Pour son exploitation, il a été réalisé un synthétiseur MIDI polyphonique basique pouvant être piloté par tout appareil équipé de la connectique MIDI sérielle à connecteur DIN5 (p.ex : claviers, séquenceurs, ordinateurs, etc.). Pour cela, un CPU softcore se charge de la conversion des événements MIDI en commandes pour le SPU.

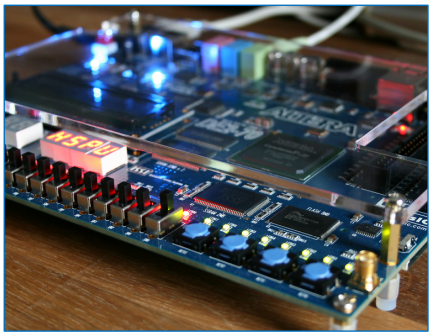
Le HSPU est actuellement capable de rendre jusqu'à 256 canaux audio simultanément à 88.2kHz en 32bits stéréo avec une très faible latence.

Travail de diplôme
| édition 2012 |

Filière
Systèmes industriels

Domaine d'application
Infotronics

Professeur responsable
Dr. Pierre-André Mudry
pierre-andre.mudry@hevs.ch



Le projet a été réalisé sur un kit de de développement embarquant une FPGA, des mémoires RAM/FLASH, un convertisseur D/A et diverses entrées/sorties.

Le HSPU peut être piloté par un clavier, un ordinateur ou tout autre appareil équipé de la connectique MIDI.

HES-SO Valais
Route du Rawyl 47
1950 Sion

Tél. 027 606 85 11
URL www.hevs.ch

TABLE DES MATIERES

1. INTRODUCTION 3
1.1 ORIENTATION DU PROJET 3
1.2 UTILISATION CIBLE 4
2. SYNTHETISEUR – ANALYSE 5
2.1 SYNTHÉTISEUR : DÉFINITION 5
2.2 A PROPOS DU TYPE DE SYNTHÉTISEUR 5
2.3 CHOIX PRÉLIMINAIRES IMPORTANTS 6
2.3.1 Détermination de l'interface de communication 6
2.3.2 Détermination du format de travail du SPU 7
2.3.3 Etude de synthétiseurs 7
2.3.3.1 Synthétiseurs software orientés MIDI 8
2.3.3.2 Synthétiseurs software orientés 'Module' 9
2.3.3.3 Synthèse 9
2.3.4 Détermination d'une board de développement 10
2.3.5 Choix du softcore 11
3. CONCEPTION 13
3.1 ARCHITECTURE FONCTIONNELLE 13
3.2 PARTIE SPU 14
3.2.1 Accès à la mémoire de travail 16
3.2.1.1 Contrôleur SSRAM 16
3.2.1.2 Multiplexeur d'accès à la mémoire 17
3.2.2 Unités de traitement sonore 18
3.2.2.1 Principe de communication entre les unités 20
3.2.2.2 Unité « Position computer » 21
3.2.2.3 Unité « Sample collector » 23
3.2.2.4 Unité « Interpolator » 24
3.2.2.5 Unité « Amplitude modifier » 24
3.2.2.6 Unité « Accumulator » 24
3.2.3 Accès aux registres et à la mémoire de travail du SPU 25
3.2.3.1 Interface Avalon 25
3.2.3.2 Registres de configuration/contrôle 26
3.2.3.3 Registres des canaux logiques 27
3.2.3.4 Accès à la mémoire de travail du SPU 28
3.2.4 Manager 28
3.2.4.1 Exclusion mutuelle 29
3.2.5 A propos du convertisseur D/A 30
3.3 PARTIE CPU 31
3.3.1 Configuration hardware 31
3.3.2 Réalisation software 34
3.3.2.1 Programme de test #1 – Synthétiseur soft « Sawtooth » 34
3.3.2.2 Programme de test #2 – Hardware « Square generator » 36
3.3.2.3 Synthétiseur MIDI « SquareHarp » 37
4. VERIFICATION 41
4.1 TESTS UNITAIRES DU SPU 41
4.1.1 Unités de traitement sonore du canal physique 41
4.1.2 Contrôleur SSRAM 42
4.2 TESTS FONCTIONNELS DU SOFTCORE, DE L'INTERFACE MIDI ET DU CODEC AUDIO 43
4.3 TESTS FONCTIONNELS DE BASE DU SPU 44
4.4 TEST GLOBAL DU SYNTHETISEUR MIDI « SQUARE HARP » 47
5. AMELIORATIONS FUTURES 49
5.1 DANS UN FUTUR PROCHE 49
5.1.1 Unité « DAHDSR Enveloppe Generator » 49

5.1.2 Unité « Response Converter » 53
5.1.3 Synchronisateur d'unités 56
5.2 DANS UN FUTUR UN PEU PLUS LOINTAIN 57
5.2.1 Une accélération plus complète du contrôle du volume 57
5.2.2 Problématique des vibratos et des portamentos 57
5.2.3 Filtres 58
5.2.4 Formats de « compression » simples 58
5.2.5 Interpolation 58
5.2.6 Nombre de sorties audio générique 58

6. SYNTHESE 59

7. CONCLUSION 60

8. DATE ET SIGNATURE 60

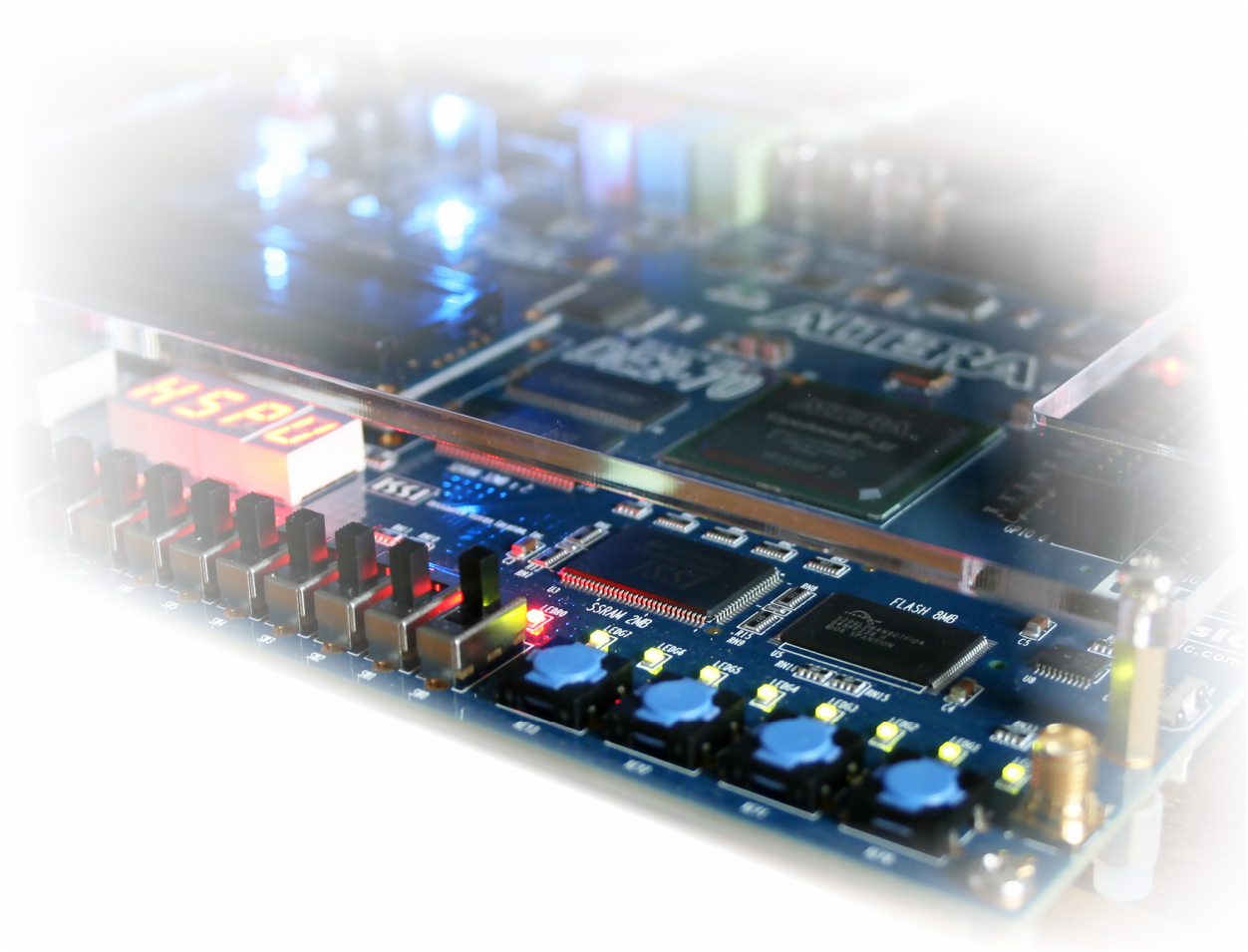
9. GLOSSAIRE 61

10. TABLE DES ILLUSTRATIONS 62

11. BIBLIOGRAPHIE 63

12. DOCUMENTS ANNEXES 64

13. CD-ROM 64



1. INTRODUCTION

De nos jours, nombreux sont les appareils ayant besoin de produire du son. Cependant, la production de son ne nécessite pas les mêmes ressources pour toutes les applications. Certains appareils, tels que les baladeurs et les téléphones portables, ont besoin d'être spécialisés dans le décodage de flux audio compressé. D'autres, comme les synthétiseurs, les consoles de jeux et les ordinateurs doivent souvent être capables de traiter une grande quantité de sons simultanés.

Le but de ce projet consiste à concevoir une unité de traitement sonore (*Sound Processing Unit*, SPU) en hardware en vue de remplir la fonction de synthétiseur basé sur la lecture d'échantillons audio (échantillonneur, *sampler*) stockés en mémoire. Sur cette base, l'utilisateur pourrait ainsi commander leur lecture à différentes fréquences et amplitudes.

Cependant, un tel SPU, seul, n'est pas en lui-même un synthétiseur, mais l'une de ses composantes principales : en tant qu'accélérateur hardware, il doit être piloté par un processeur (*Central Unit Processing*, CPU). Ainsi, le SPU sera intégré dans un tout réalisant la fonction de synthétiseur MIDI.

Ce rapport vous présentera tout d'abord une courte étude de synthétiseurs MIDI existants pour évaluer les fonctions qu'un tel système devrait présenter. Ensuite, vous sera proposé une architecture hardware du SPU avec son implémentation détaillée, suivi du développement de plusieurs codes de démonstration pour le processeur le pilotant. Finalement, ses résultats seront analysés et discutés.

1.1 ORIENTATION DU PROJET

Le type de SPU devant être réalisé dans ce projet ne peut trouver place que dans des systèmes nécessitant de la lecture massivement parallèle d'échantillons sonores. On pourrait identifier trois grandes catégories d'utilisation cible :

- **Usage pour interface homme-machine**
  Dans le cas de certaines interfaces homme-machine, le son peut occuper une place non négligeable : si celles-ci ne se contentent pas de lire de temps à autre de simples sons préenregistrés, il peut devenir intéressant (but d'économie de ressources CPU), voir obligatoire (CPU pas suffisamment performant, ceci concerne essentiellement des systèmes embarqués) d'accélérer matériellement les traitements audio.
- **Usage à but musical**
  Il s'agit du domaine d'utilisation principal nécessitant l'emploi de SPU. Les instruments électroniques offrent beaucoup de liberté dans la création et le jeu musical. Voici trois avantages des plus pertinents :
  - Synthèse de sons très difficilement/impossiblement reproductibles par des moyens non électroniques.
  - Possibilité de reproduire une grande quantité d'instruments réels enregistrés… mais leur encombrement en moins : très portable.
  - Simplification et ouverture gigantesque à la création musicale.
- **Usage générique/mixte**
  Certains appareils peuvent avoir besoin de séquencer des partitions musicales tout en produisant des bruitages, et ce, généralement à des fins de divertissement. Cela concerne essentiellement les PC et les consoles de jeux.

Sur la base de ces orientations possibles, la réalisation de ce projet a été axée sur la finalité d'un usage musical, car celui-ci est exigeant en ce qui concerne les fonctionnalités requises et les performances. Ainsi, ce choix rend ce travail utilisable pour tout autre usage.

1.2 UTILISATION CIBLE

Un tel SPU trouve son usage principalement dans les appareils cibles suivants :

- **Racks synthétiseurs**
  Utilisés en studio ou en live, ces racks sont contrôlés par un, voire plusieurs claviers MIDI maîtres (sans aucun module de rendu sonore) ou d'autres périphériques (batteries électroniques, séquenceurs, ordinateurs, etc…). Ils peuvent contenir des synthétiseurs de type et de synthèse sonore différents, bien que la synthèse par lecture d'échantillons soit très répandue.
- **Claviers synthétiseurs**
  Dans ce cas, le synthétiseur est simplement incorporé au clavier. Les claviers à la norme MIDI nécessitent en général l'emploi d'un tel processeur, puisqu'ils utilisent très souvent des banques sonores échantillonnées.

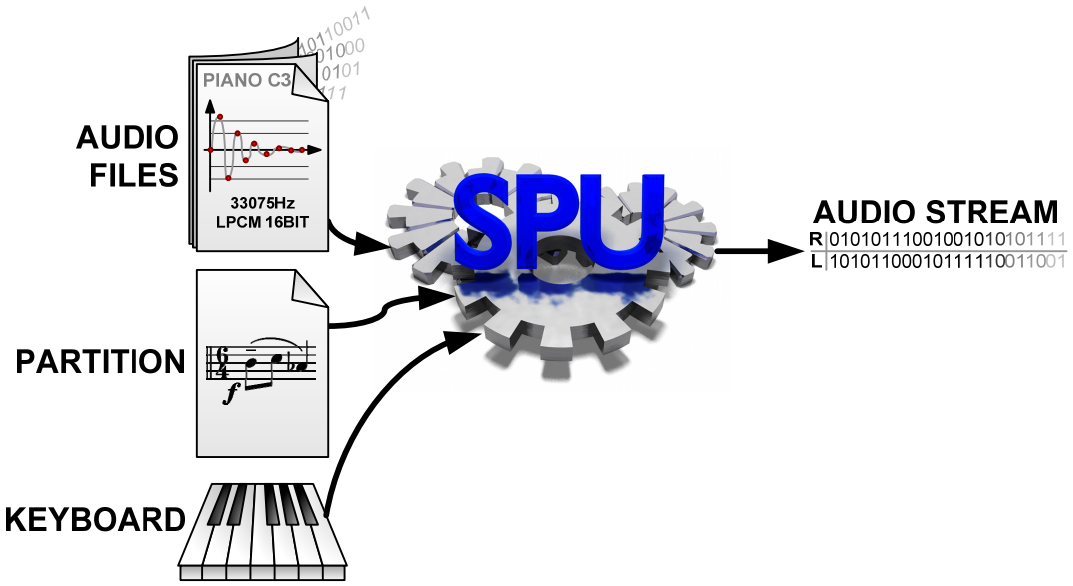


Fig. 1 – Concept général du SPU

2. SYNTHÉTISEUR – ANALYSE

2.1 SYNTHÉTISEUR : DÉFINITION

Un synthétiseur est un appareil musical étudié pour créer/générer des sons électroniques au moyen de différentes techniques, analogiques ou numériques. Pour cela, des signaux (issus d'oscillateurs, de tables d'ondes, d'échantillons, etc…) sont manipulés en fréquence, amplitude, en clarté (filtres), additionnés/soustraits à d'autres signaux, etc… afin de produire un instrument musical[1].

2.2 A PROPOS DU TYPE DE SYNTHÉTISEUR

Avec le traitement numérique du son, les possibilités pour (re)produire un instrument sont grandes : cela passe de l'émulation stricte de circuits analogiques (synthèse additive / soustractive / FM, etc…) à la lecture d'échantillons issus d'enregistrements (*samplers*).

Les synthétiseurs numériques basés sur la reproduction de circuiterie analogique, émulent essentiellement des oscillateurs qui interagissent sur le son final en passant par diverses fonctionnalités 'mathématiques'. La figure suivante présente un exemple basique de synthétiseur analogique :

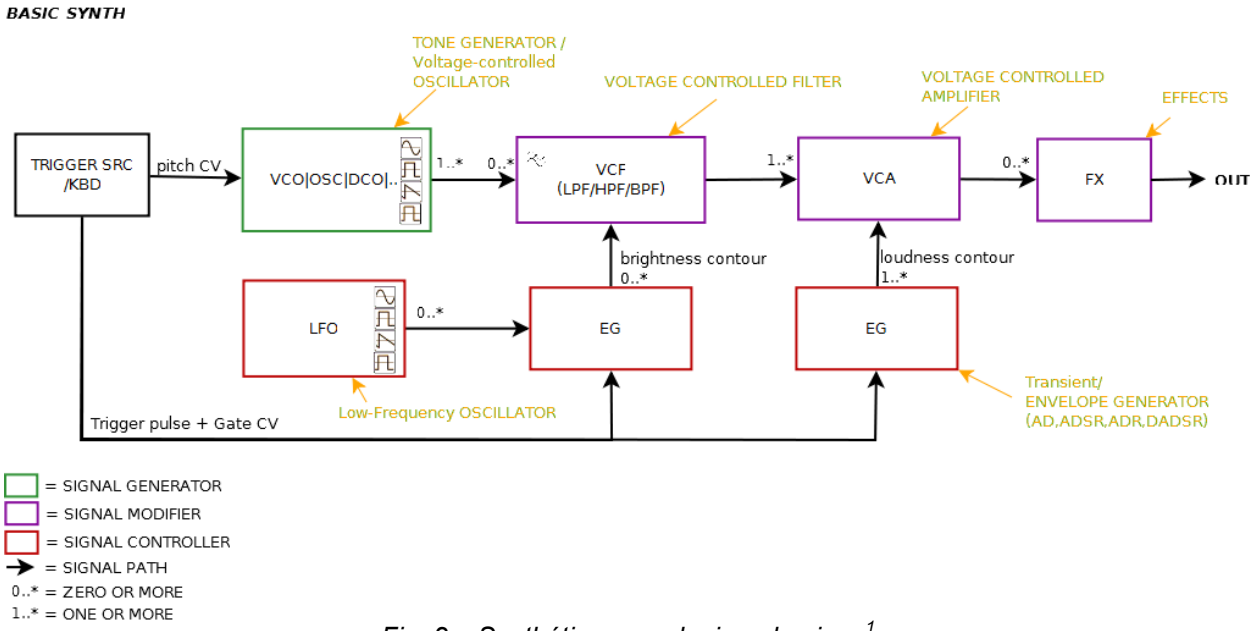


Fig. 2 – Synthétiseur analogique basique[1]

Un synthétiseur basé sur un *sampler* ne fera que remplacer le bloc « TONE GENERATOR » et une partie des autres oscillateurs par des lecteurs d'échantillons stockés en mémoire. Cela donne l'opportunité de pouvoir remplir les fonctionnalités des synthétiseurs à oscillateurs, tout en y rajoutant la possibilité de pouvoir reproduire des instruments réels.

L'implémentation d'une telle solution pour le SPU est intéressante, puisqu'elle offre non seulement la possibilité de l'utiliser à des fins musicales, mais aussi pour de la lecture de bruitages.

[1] Source / référence : *http://fr.wikipedia.org/wiki/Synthétiseur*

NB : Formellement, un système effectuant un rendu sonore uniquement basé sur de la lecture d'échantillons stockés en mémoire et ce, sans altération, ne s'appelle pas « synthétiseur », mais « *sampler* (échantillonneur) ». Cependant, le seuil entre synthétiseur et *sampler* n'est pas vraiment très clair : le fait de rééchantillonner (changer la hauteur) et d'altérer l'amplitude pourrait déjà être qualifié de synthétiseur. Aussi, cette distinction perd de son sens selon les cas d'utilisation. En effet, il est possible de faire de la synthèse additive avec un *sampler* en utilisant des fondamentales échantillonnées plutôt que générées par d'autres processus. Pour ne pas troubler le lecteur, ces deux termes ne seront pas distingués ; ainsi, par abus de langage, l'appellation « synthétiseur » fera foi dans ce rapport.

2.3 CHOIX PRÉLIMINAIRES IMPORTANTS

Il n'existe pas vraiment de normes à respecter pour le comportement d'un SPU. Cependant, sa structure déterminera grandement sa simplicité d'utilisation et son champ d'application finale. Ainsi, une recherche a dû être effectuée pour déterminer les fonctionnalités les plus couramment utilisées dans le domaine musical.

Bien sûr, un SPU est strictement inutile s'il ne peut être piloté par un clavier, un séquenceur ou tout autre appareil : c'est pour cela que le choix d'une interface de communication a également été étudié.

2.3.1 DÉTERMINATION DE L'INTERFACE DE COMMUNICATION

Le SPU a implicitement besoin d'être piloté par un CPU et puisqu'il est étudié pour du rendu musical, il serait très intéressant qu'il puisse être lui-même piloté par un clavier ou un ordinateur, ce qui est facilement possible avec la norme MIDI[2].

MIDI est un protocole très répandu de communication entre instruments de musique électroniques (p.ex. claviers, ordinateurs, etc…). Sa réalisation datant du début des années 80, il est relativement simple à implémenter puisque basé sur une communication sérielle sur laquelle transitent des événements musicaux (p.ex.: début/fin de note jouée, altération de la hauteur (pitch), etc…). La figure ci-contre présente la connectique MIDI.

Avec le temps, le protocole, particulièrement flexible, a évolué tout en restant compatible avec les fonctionnalités de base. Aussi, la connectique DIN5[3] (qui lui était associée à la base) a tendance à être complétée, voire remplacée par de l'USB ou du IEEE1394 (FireWire)[4] quand il s'agit de relier un instrument à un ordinateur.

Fig. 3 – Connectique DIN5 utilisée pour le protocole MIDI

Highlights :

- Communication sérielle monodirectionnelle (source de courant)
  - Start bit : 1
  - Data : 8bits
  - Parity : None
  - Stop bit : 1
  - 31'250 Bauds/s
- Connecteur DIN à 5 broches

[2] Réf. : *http://www.midi.org/techspecs/* , *http://fr.wikipedia.org/wiki/Musical_Instrument_Digital_Interface*
[3] Réf. : *http://www.midi.org/techspecs/electrispec.php*
[4] Réf. : *http://www.midi.org/aboutmidi/tut_midicables.php*

Puisque, malgré son âge, MIDI est toujours utilisé sur les claviers, PC, etc…, il a été retenu dans sa version DIN5 à but démonstratif.

2.3.2 DÉTERMINATION DU FORMAT DE TRAVAIL DU SPU

Le SPU devra manipuler des échantillons audio stockés en mémoire. Néanmoins, pour des raisons de simplicité et de performance, l'utilisateur du SPU ne pourra lui fournir que des échantillons audio non compressés (en *integer* – LPCM). L'utilisation de la virgule flottante n'est pas considérée, car elle est particulièrement lourde en ressources.

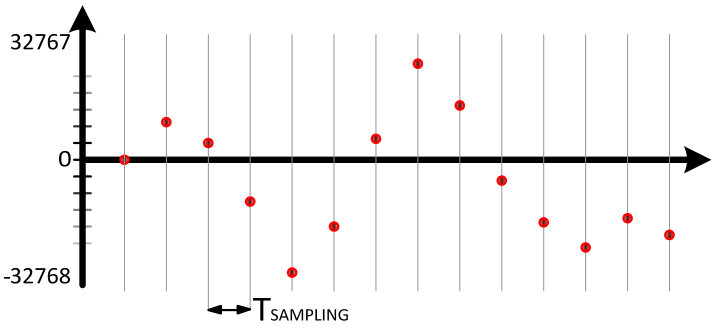


Fig. 4 – Représentation d'un échantillon audio non compressé (LPCM, 16bits signé)

Il faut également savoir qu'il serait dommage de se limiter à un format précis en particulier (résolution, signe et *endianness* fixes), ne serait-ce qu'à cause des considérations suivantes :

- Les échantillons sont en général stockés en 16bits (signés, little endian).
- De nombreux vieux échantillons utilisés sont en 8bits (généralement non signés).
- Pour des raisons de compression des banques sonores dans des cas spécifiques (typiquement : manque d'espace mémoire), certains échantillons sont réduits à 8bits.
- L'utilisation de résolutions supérieures à 16bits dans les enregistrements récents a fait apparaître l'utilisation d'échantillons typiquement en 24bits (*integer*) et en 32bits (flottant).

Pour cela, les caractéristiques suivantes sont retenues :

- **Bit depth** : *8 / 16 / 32 bits**
- **Sign** : *Unsigned / Signed*
- **Endianness** : *Big endian, Little endian*

* Ces tailles sont adaptées pour un usage sur une mémoire 32bits. Les échantillons seront ainsi alignés (une case mémoire 32bits → 4 échantillons à 8bits, 2 échantillons à 16bits ou 1 échantillon à 32bits).

Aussi, la fréquence d'échantillonnage générale f_{MIXING} (c'est-à-dire la fréquence à laquelle les échantillons rendus par le SPU sont débités en sortie) ne devrait pas être limitée à une valeur fixe et devrait au moins couvrir 22.5kHz à 96kHz (standard).

2.3.3 ETUDE DE SYNTHÉTISEURS

Pour éviter de produire un SPU ne faisant rien d'autre que d'accélérer les fonctionnalités spécifiques d'un synthétiseur software ou hardware déjà existant (en bref : une pâle copie), quelques synthétiseurs softwares libres/gratuits/payants répandus ont été étudiés afin de relever les points forts de chacun, ainsi que leurs points communs.

2.3.3.1 SYNTHÉTISEURS SOFTWARE ORIENTÉS MIDI

Sur PC, lorsque la lecture d'un fichier MIDI est lancée sur un lecteur multimédia ou sur un logiciel de composition musicale (MAO), le synthétiseur software utilisé reçoit les événements MIDI et effectue le rendu sonore. Bien que la norme MIDI ait attribué un grand nombre d'événements à des fonctions spécifiques, ces événements sont implémentés (tant en fonctionnalité qu'en comportement) selon le bon vouloir de l'auteur du synthétiseur. Néanmoins, il existe des fonctionnalités minimales communes qui ont pu être relevées par l'étude des synthétiseurs suivants :

- **Yamaha SoftSynthesizer S-YXG50** *(Windows)*
  Banque de 2 ou 4MB (compressée, format propriétaire), ~1996-2001, plus mis à jour depuis ~2002.
- **Creative SoundBlaster Synth** *(multiples déclinaisons, Windows)*
  Accélération hardware – soit basée sur une ROM physique, soit sur des banques au format SoundFont[5] V1 ou V2.
- **Microsoft GS Wavetable Software Synth** *(Windows)*
  Présent par défaut dans tous les Windows (synthé principal).
  Basé sur une banque Roland™ GM/GS 4MB au format DLS[6] – 1996.
- **FluidSynth**[7] *(Linux/Multi-plateformes, opensource)*
  Charge des banques au format SoundFont V2.
- **ZynAddSubFX**[8] *(Multi-plateformes, opensource)*
  Basé sur de la synthèse additive/soustractive. Projet créé en 2002.

Voici les points communs :

Note Event Level
▪ Note ON/OFF – with note amplitude & corresponding channel
▪ Pitch modifier
MIDI Channel Level
▪ Volume (logarithmic behaviour)
▪ Panning (logarithmic behaviour)
▪ Instrument selection
▪ Frequency modulation
Audio render specific
▪ ADSR amplitude modulation envelope

Ces caractéristiques communes sont toutes définies dans la norme General MIDI GM1[9] à laquelle tous ces synthétiseurs adhèrent (à l'exception de ZynAddSubFX).

Voici également quelques fonctionnalités intéressantes spécifiques à certains d'entre eux (essentiellement YAMAHA Y-SXG50, FluidSynth et ZynAddSubFX) :

[5] Infos complémentaires : *http://connect.creativelabs.com/developer/SoundFont/Forms/AllItems.aspx*
[6] Infos complémentaires : *http://www.midi.org/techspecs/dls/dls1v11b.pdf*
[7] Réf. : *http://*www.*fluidsynth.org/*
[8] Réf. : *http://zynaddsubfx.sourceforge.net/*
[9] Réf. : *http://www.midi.org/techspecs/gmguide2.pdf*
MIDI Manufacturers Association, General MIDI System Level 1 Developer Guidelines for Manufacturers and Composers, Second Revision, July 1998

MIDI Channel Level

- Reverberation (customizable)
- Chorus / Flanger
- Others various effects (Distortion, Delay, Wah, etc…)
- Resonant low pass filter (order 2)
- Low/High pass filters (different types)
- …

Elles présentent toutes l'avantage de rendre le son moins « linéaire », « parfait », ce qui lui apporte une grande richesse.

2.3.3.2 SYNTHÉTISEURS SOFTWARE ORIENTÉS 'MODULE'

Les modules sont des fichiers contenant non seulement la partition, mais aussi les échantillons/instruments permettant une reproduction fidèle et indépendante de l'ordinateur/lecteur qui les lit.
A l'exception de quelques formats de modules récents, leur fonctionnement est très orienté 'hardware' : on y compose dans des canaux qui peuvent directement être transposés/joués en hardware (si possible) sans grands traitements.

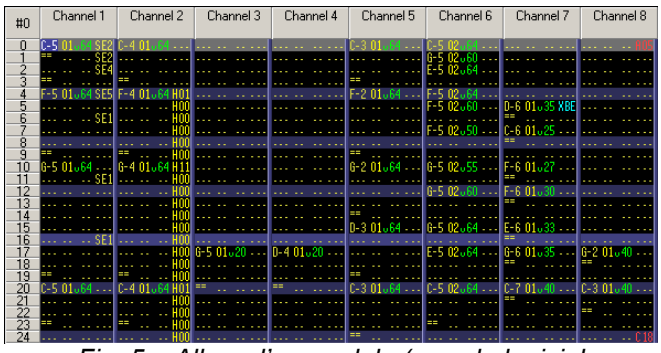


Fig. 5 – Allure d'un module (sous le logiciel OpenMPT[10])

Chaque canal ne se contente pas seulement de lire des échantillons sans y appliquer la moindre altération. La liste suivante résume les fonctionnalités offertes par le format de module FastTracker [*.xm] (regroupant celles de nombreux formats plus anciens) :

- On samples : Loop (mono/bidirectional), auto-vibrato
- On instruments : Volume, pan & vibrato envelope
- On channels : Volume (Including : slide, vibrato, tremolo)
- On channels : Panning (Including : slide)
- On channels : Tone alteration (Including : Portamento/Glissando, Arpeggio, Fine tune)
- On channels : Sample playing control (Including : Delay, Start offset)

2.3.3.3 SYNTHÈSE

Les divers synthétiseurs précédemment analysés sont construits autour d'une certaine base commune, c'est-à-dire la norme GM1.
En ce qui concerne le pilotage du SPU par un périphérique MIDI, il a été relevé que le respect de cette norme serait quasi nécessaire (sans tenir compte de certains points comme la liste d'instruments imposés et certains contrôles « superflus »).

Pour ce qui est du comportement de ces synthétiseurs, les points suivants sont retenus :

Au niveau du contrôle direct :

- Contrôle direct du volume et du panning*
- Altération directe de la hauteur (pitch)*
- Note ON/OFF*

Au niveau traitement :

- Modulation d'enveloppe sur le volume et sur la hauteur (pitch)
- Au moins une boucle pour la lecture d'échantillons*
- Filtre passe-bas d'ordre 2 résonant *(éventuellement contrôlé par modulation d'enveloppe)*
- Filtre passe-haut *(éventuellement contrôlé par modulation d'enveloppe)*

** Eléments basiques à implémenter au minimum.*

[10] Logiciel de ModuleTracker open source, *http://openmpt.org/*

2.3.4 DÉTERMINATION D'UNE BOARD DE DÉVELOPPEMENT

Le HSPU doit être implémenté dans une FPGA reliée aux ressources suivantes :

- Un D/A audio de bonne qualité (min. 44.1kHz 16bits stéréo).
- Une mémoire RAM de taille supérieure à 1MBytes pour le stockage des échantillons.
- Quelques LEDs pour l'indication de l'état du HSPU.

Le HSPU doit également être piloté par un CPU afin de l'utiliser comme synthétiseur MIDI. Le choix du CPU doit tenir compte des points suivants :

- Puissance de calcul suffisante pour la conversion des événements MIDI en instructions pour le HSPU.
- UART pour la connectique MIDI.
- UART pour du debugging.
- LEDs et switches pour du contrôle/debugging.
- Accès à une mémoire FLASH pour le stockage d'une banque sonore.

A notre disposition se trouvaient divers kits équipés de FPGA de la marque Xilinx ou Altera. Pour des raisons d'habitude de travail avec ses outils de développement et pour son riche contenu, la board de développement DE2-70 de Terasic[11] a été retenue.

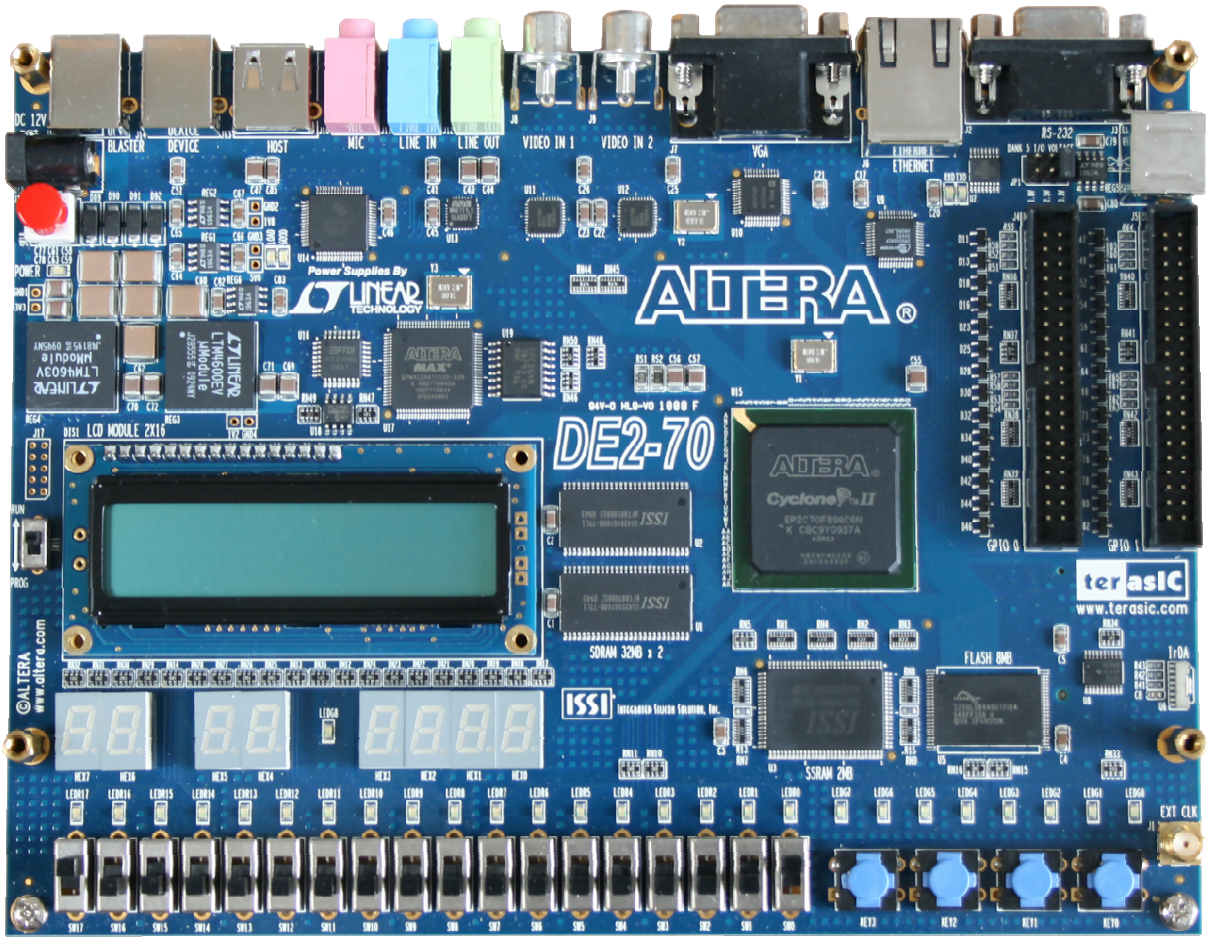


Fig. 6 – Kit de développement FPGA «Terasic DE2-70»

[11] Site du fabricant : *http://www.terasic.com.tw/en/*

Elle est entre autre équipée des composants suivants :

- FGPA « Altera EP2C70F896C6N »
  68'416 LEs, 4 PLL, 250 Memory blocks (4kbits), 150 Multipliers 18x18
- Codec audio « Wolfson WM8731 »
  24bit, 8kHz to 96kHz, including line in and microphone inputs
- 1x SSRAM (512K x 36bits / 2Mbytes)
- 2x SDRAM (16M x 16bits / 32Mbytes)
- 1x PARALLEL FLASH (8M x 8bits / 8Mbytes)
- Misceanellous
  RS232, PS2, IRDA, VGA 24bits, Ethernet, TV decoder, LCD 2x16chars, 7-segments, LEDs, switches, buttons,…

Ayant déjà codé dans le passé un bloc VHDL pour le contrôle du codec audio, celui-ci a pu être utilisé 'tel quel' pour économiser du temps.

2.3.5 CHOIX DU SOFTCORE

Altera propose également des softcores du nom de Nios*II*. La mise en service d'un ou plusieurs de ces CPU se réalise au travers de l'outil « QSys », intégré à l'IDE Quartus*II*. L'utilisation d'une telle solution permet d'éviter l'emploi d'une seconde board de développement avec CPU. Elle permet également de rapidement mettre en œuvre un ou plusieurs softcores, avec les IP d'accès aux mémoires SDRAM, SSRAM, FLASH ainsi que diverses entrées/sorties (UART, GPIO, etc...), et ce, essentiellement graphiquement. C'est pour cela que cette solution a été retenue.

Les softcores Nios*II* existent sous trois déclinaisons, de la plus compacte à la plus performante. Voici un extrait de leurs datasheets :

Feature		Core		
		Nios II/e	Nios II/s	Nios II/f
Objective		Minimal core size	Small core size	Fast execution speed
Performance	DMIPS/MHz *(1)*	0.15	0.74	1.16
	Max. DMIPS *(2)*	31	127	218
	Max. f_{MAX} *(2)*	200 MHz	165 MHz	185 MHz
Area		< 700 LEs; < 350 ALMs	< 1400 LEs; < 700 ALMs	Without MMU or MPU: < 1800 LEs; < 900 ALMs With MMU: < 3000 LEs; < 1500 ALMs With MPU: < 2400 LEs; < 1200 ALMs
Pipeline		1 stage	5 stages	6 stages
External Address Space		2 GB	2 GB	2 GB without MMU 4 GB with MMU
Instruction Bus	Cache	–	512 bytes to 64 KB	512 bytes to 64 KB
	Pipelined Memory Access	–	Yes	Yes
	Branch Prediction	–	Static	Dynamic
	Tightly-Coupled Memory	–	Optional	Optional
Data Bus	Cache	–	–	512 bytes to 64 KB
	Pipelined Memory Access	–	–	–
	Cache Bypass Methods	–	–	■ I/O instructions ■ Bit-31 cache bypass ■ Optional MMU
	Tightly-Coupled Memory	–	–	Optional
Arithmetic Logic Unit	Hardware Multiply	–	3-cycle *(3)*	1-cycle *(3)*
	Hardware Divide	–	Optional	Optional
	Shifter	1 cycle-per-bit	3-cycle shift *(3)*	1-cycle barrel shifter *(3)*
JTAG Debug Module	JTAG interface, run control, software breakpoints	Optional	Optional	Optional
	Hardware Breakpoints	–	Optional	Optional
	Off-Chip Trace Buffer	–	Optional	Optional

*Fig. 7 – Extrait des caractéristiques des différents softcores Nios*II*[12]*

Pour un bon confort de développement, la version Nios*II*/f a été retenue. Il est à noter que les versions Nios*II*/s et Nios*II*/f sont payantes. Cependant, elles peuvent être utilisées gratuitement – sans toutefois pouvoir programmer la FLASH de la FPGA avec. La transition d'une version à une autre peut se faire aisément, sans conséquences particulières sur le code. L'environnement de développement, également fourni par Altera, n'est rien d'autre que Eclipse[13] (très répandu).

[12] Réf. : NiosII Processor Reference Handbook, *www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf, Page 5-1*
[13] Infos complémentaires : *www.altera.com/devices/processor/nios2/tools/ide/ni2-ide.html* et *www.eclipse.org*

3. CONCEPTION

Le système se présente sous la forme d'une carte sur laquelle l'utilisateur peut brancher un périphérique MIDI. La banque sonore (comprenant les échantillons audio, leur mapping sur les instruments et les paramètres des instruments) est interprétée selon les événements MIDI reçus dans le but de générer les commandes pour le SPU.

Ainsi, l'utilisateur peut contrôler le système soit directement par un clavier, soit par un séquenceur ou par un autre appareil MIDI, comme résumé dans la figure suivante :

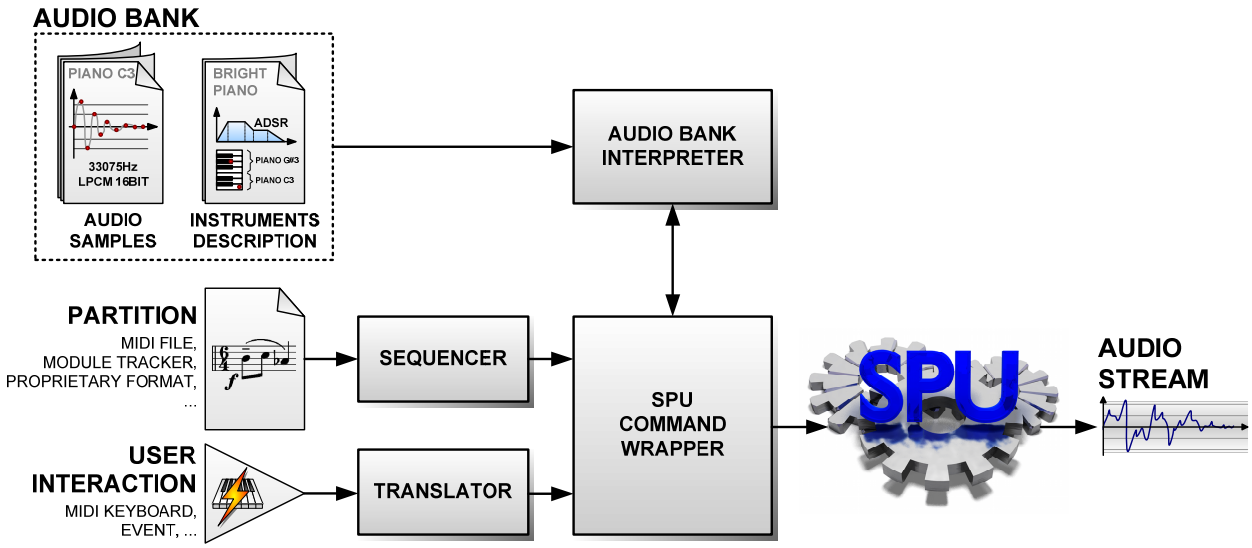


Fig. 8 – Schéma bloc d'utilisation du SPU

3.1 ARCHITECTURE FONCTIONNELLE

Le système est décomposable en deux parties centrales :

- **Le SPU**
  Produit le son sur la base de la configuration de ses registres.
- **Le CPU**
  Doit piloter le SPU sur la base de la banque sonore et des événements MIDI qu'il reçoit.

Pour son exploitation, le système a besoin d'être piloté par un ou plusieurs appareils MIDI (ceci nécessite l'ajout de connecteurs) et d'être relié à un amplificateur ou autre consommateur sonore.

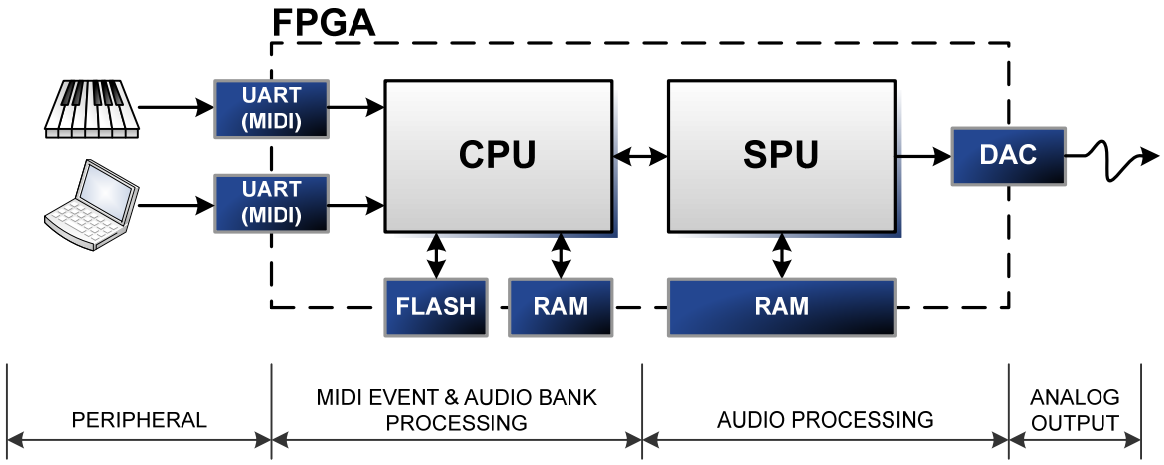


Fig. 9 – Architecture générale

3.2 PARTIE SPU

Le SPU possède une structure par canaux. Lorsque l'utilisateur veut lire un son en mémoire (p.ex. une note d'un piano), il doit paramétrer un canal avec les informations adéquates :

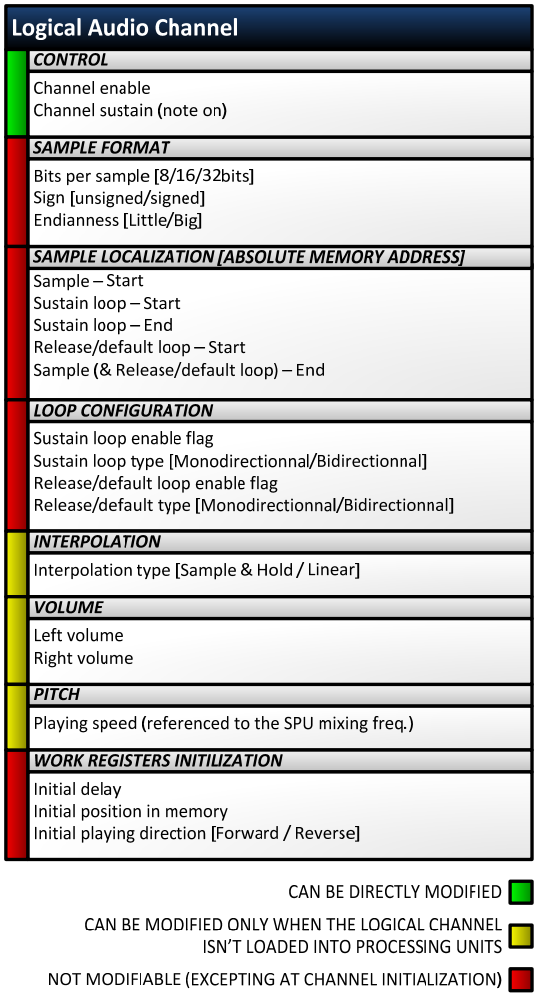


Fig. 10 – Canal audio logique

Cependant, ce canal ne correspond pas à un canal physique dans la FPGA : on parle ainsi de **canal logique**. Le SPU est apte à contenir une certaine quantité de ces canaux logiques qu'il copiera tour à tour dans le canal physique composé des unités de traitement (méthode du round robin).

Pour chacun de ces canaux logiques seront calculés '*n*' (défini par l'utilisateur) échantillons de sortie qui seront accumulés pour former un **paquet** une fois le tour des canaux logiques effectué. Chaque paquet complet est transmis dans le buffer (FIFO) du D/A. Plus la taille des paquets est grande, plus le système aura de latence (sera abordé ultérieurement au point 4.4).

Ce fonctionnement par canaux a abouti à une structure dont voici la forme générale :

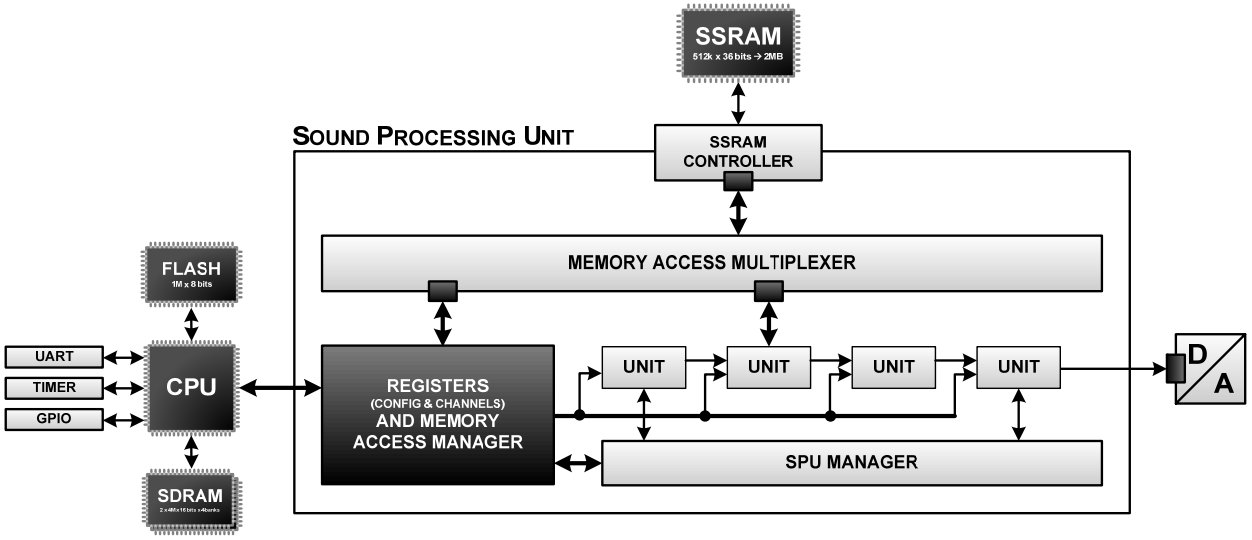


Fig. 11 – Structure générale du système

Tel que présenté, le SPU peut être scindé en quatre parties principales :

- **Memory Access** *(regroupe « Memory Access Multiplexer » et « SSRAM Controller »)*
  Les blocs ayant besoin d'accéder à la mémoire de travail du SPU utilisée pour le stockage des échantillons audio, passent par un multiplexeur d'accès assurant que tous les accesseurs soient servis. Ce multiplexeur interface directement le contrôleur SSRAM.
- **Unit Chain** *(regroupe toutes les unités de traitement sonore)*
  La chaîne réalisée par une section d'unité forme le canal physique de rendu sonore dans lequel seront chargés tour à tour les canaux logiques.
- **Registers and Memory Access Manager**
  Ce bloc contient les registres de contrôle du SPU, ainsi que ceux des canaux logiques. Du côté CPU, il fait office de bridge, puisqu'il lui permet un accès aux registres et à la mémoire de travail. Du côté SPU, il place les registres du canal logique sélectionné par le « Manager » sur les unités du canal physique. Il fournit également les registres de configuration/contrôle au « Manager ».
- **SPU Manager**
  Le « Manager » séquence l'activité du SPU, à savoir : placement des canaux logiques dans le canal physique, surveillance du rendu des canaux logiques, transmission du paquet rendu au D/A. Ces opérations sont cycliques.

Le SPU a été étudié pour rendre jusqu'à 256 canaux logiques avec une granularité des paquets comprise entre 1 et 256 échantillons. Avec ces spécifications, un échantillon (stéréo) en sortie du SPU (sur le D/A) a le nombre de cycles d'horloge suivant pour être calculé :

$$n_{CYCLE\,PER\,OUTPUT\,SAMPLE\,COMPUTATION} = \frac{f_{CLOCK}}{f_{SAMPLING}} = \frac{50M}{88.2k} = \underline{\underline{566.9\,cycles}}$$

Et donc pour chaque échantillon calculé d'un canal logique :

$$n_{CYCLE\,PER\,SAMPLE\,COMPUTATION} = \frac{f_{CLOCK}}{f_{SAMPLING} \cdot n_{CHANNELS}} = \frac{50M}{88.2k \cdot 256} = \underline{\underline{2.214\,cycles}}$$

La structure étudiée permet d'atteindre ces performances. Pour cela, aucune des unités composant le canal physique ne prend plus de 2 cycles pour effectuer son travail. Cela laisse une petite marge pour les opérations de *context switching*, de balayement des canaux logiques, etc…

3.2.1 ACCÈS À LA MÉMOIRE DE TRAVAIL

Le fonctionnement du SPU est très orienté « *streaming* » : la vitesse de traitement est plus importante que la latence de traitement. L'accès à la mémoire a été étudié dans cette optique (voir annexe « SCH3 »).

3.2.1.1 CONTRÔLEUR SSRAM

Le SPU possède sa propre mémoire de travail. Avec les spécifications du SPU, il est possible de déterminer la bande passante nécessaire dans le pire des cas, soit avec 256 canaux logiques jouant avec des samples stockés en 32bits :

$$\begin{aligned} Débit &= f_{MIXING} \cdot sampleSize_{Byte} \cdot \text{int}\,erpolationSampleCount \cdot \log icalChannelCount \\ &= 88'200 \cdot 4 \cdot 2 \cdot 256 = 180'633'600[\text{Bytes/s}] = \underline{\underline{172.3[MBytes/s]}} \end{aligned}$$

Or, si on met le clock de la SSRAM de la board de développement à 125MHz (même valeur que pour la SDRAM du CPU), elle offre une bande passante de :

$$Débit = f_{CLOCK} \cdot memoryWidth_{Bytes} = 125MHz \cdot 4 = \underline{\underline{476.8[MBytes/s]}}$$

Cette valeur est vraie lorsque seules des lectures ou des écritures sont effectuées sans entrelacement, ce qui est le cas (le CPU charge les échantillons au démarrage dans le SPU, puis le SPU ne fait par la suite que des lectures d'échantillons).

Pour simplifier, la SSRAM est cadencée à la même fréquence que le SPU, soit 50MHz, ce qui donne :

$$Débit = f_{CLOCK} \cdot memoryWidth_{Bytes} = 50MHz \cdot 4 = \underline{\underline{190.7[MBytes/s]}} > 172.3[MBytes/s]$$

3.2.1.1.1 Implémentation

Ce contrôleur SSRAM a été réfléchi pour un accès en flux – c'est-à-dire qu'il y a une FIFO d'entrée (recevant les commandes de lectures/écritures) et de sortie (pour les données lues).

Implémenté génériquement, il prend en charge des SSRAM de largeur de données multiples de 8bits (+1bit de parité), de largeur d'adresse quelconque, un masque d'écriture (pour ne modifier que certains bytes), ainsi que le contrôle des données par parité.

Pour obtenir un débit d'accès maximal à la SSRAM, l'architecture suivante a été pensée :

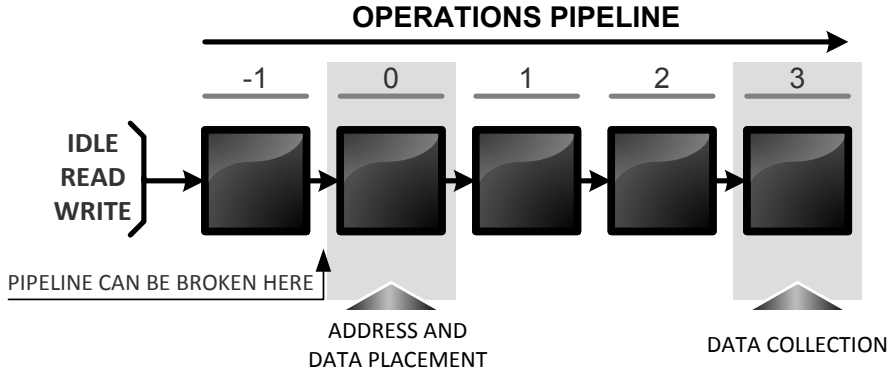


Fig. 12 – Concept d'implémentation du contrôleur SSRAM

Les commandes (lecture, écriture) sont pipelinées (registres à décalage) pour garder une trace des opérations en cours. L'opération exécutée est celle dans le registre "0". Si une lecture a été effectuée, la sortie de la SSRAM sera lue et placée dans la FIFO de sortie lorsque cette

commande de lecture aura atteint le registre "3". Toute opération d'écriture apparaissant en sortie de la FIFO d'entrée (soit "-1") sera bloquée tant qu'une opération de lecture sera présente entre "0" et "2" (sera injectée en "0" une opération « Idle »).

Plus de détails peuvent être trouvés en annexe « SCH4 – page 1 ».

3.2.1.2 MULTIPLEXEUR D'ACCÈS À LA MÉMOIRE

Ce bloc a pour objectif d'offrir un accès à une mémoire commune de manière transparente pour tous les blocs qui y sont reliés (quantité générique) par la méthode du *round robin* (taille maximum des *bursts* configurée à 16 accès). Il assure également un changement de domaine d'horloge – l'accès étant basé sur des commandes transitant à travers de FIFOs, comme pour le contrôleur SSRAM.

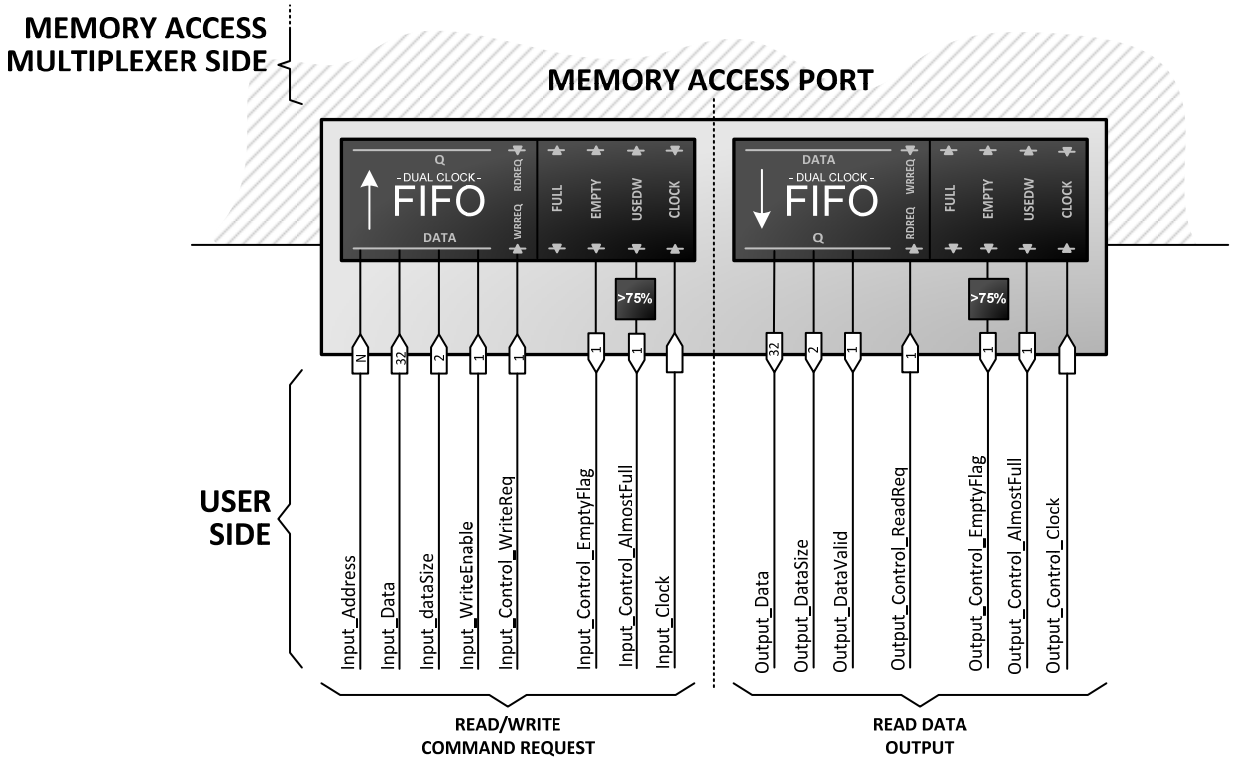


Fig. 13 – Structure d'un port du multiplexeur d'accès à la mémoire de travail

Un port d'accès se présente de la manière suivante :

Nom	*Description*
READ/WRITE COMMAND REQUEST (INPUT FIFO)	
Address	Adresse mémoire (pointant des cases mémoire 8bits)
Data	La donnée à écrire
DataSize	Taille de la donnée à écrire (8bits, 16bits ou 32bits)
WriteEnable	Demande de lecture ou d'écriture
Control_WriteReq	Contrôle de la FIFO – Ecriture de la commande
Control_EmptyFlag	Contrôle de la FIFO – FIFO vide
Control_AlmostFull	Contrôle de la FIFO – FIFO presque pleine
Clock	Horloge du bloc utilisant ce port
READ DATA OUTPUT (OUTPUT FIFO)	
Data	Donnée lue
DataSize	La taille de la donnée lue (8bits, 16bits ou 32bits)
DataValid	Indique si la parité de la donnée lue était valide
Control_ReadReq	Contrôle de la FIFO – Lecture de la donnée
Control_EmptyFlag	Contrôle de la FIFO – FIFO vide
Control_AlmostFull	Contrôle de la FIFO – FIFO presque pleine
Clock	Horloge du bloc utilisant ce port

Tabl. 1 – Signaux d'un port du multiplexeur d'accès à la mémoire de travail

Pour plus d'informations quant à son implémentation, veuillez-vous référer à l'annexe « SCH4 – Page 2 ».

Ainsi, entre les différentes FIFO et latences de ce bloc et celles du contrôleur SSRAM, un accès à la mémoire prend en moyenne une dizaine de cycles – entre le moment où une commande de lecture est placée dans la FIFO d'entrée d'un port et l'apparition de la donnée dans la FIFO de sortie. Cette latence est réduite de moitié pour une écriture.

3.2.2 UNITÉS DE TRAITEMENT SONORE

La production du son est réalisée par une chaîne de petites unités ayant chacune un but bien précis. Chaque unité est configurée par des propriétés définies par l'utilisateur (via les registres des canaux logiques) afin que l'ensemble puisse faire le rendu d'un canal audio logique. Pour pouvoir produire '*n*' sons en parallèle, cette chaîne subit des changements de contexte pour chaque '*m*' échantillons rendus (taille d'un paquet).

Le rendu sonore y est effectué sur le principe du 'streaming' : une fois un ordre de calcul lancé à la première unité (Position Computer), son résultat est directement transmis à l'unité suivante. Cette unité consommera aussitôt ce résultat et passera le sien à la suivante, et ainsi de suite… jusqu'à que le résultat final parvienne à la dernière unité (Accumulator). Pour produire des paquets de taille '*m',* le calcul d'un échantillon de sortie est initié '*m*' fois.

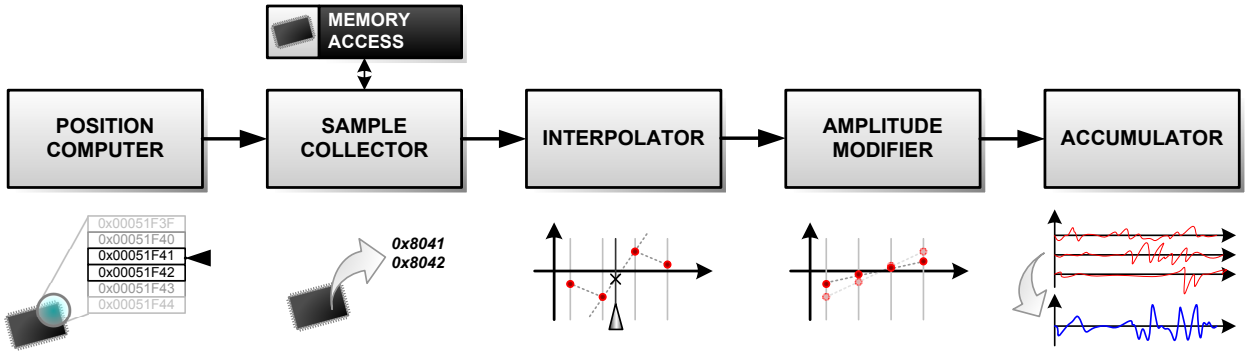


Fig. 14 – Canal physique de rendu sonore réalisé par le chaînage des unités de traitement sonore

Les fonctions principales réalisées par cette chaîne sont les suivantes :

Tout d'abord, l'unité « Position Computer » détermine la position des échantillons (deux pour une interpolation linéaire) à collecter dans la mémoire à un temps donné afin d'obtenir la hauteur de son désirée. La vitesse de déplacement en mémoire est donnée par une vitesse de lecture des échantillons (*Fig. 15*). Ces calculs sont effectués à virgule fixe.

Ensuite, le « Sample Collector » se charge de collecter les échantillons nécessaires en mémoire et les convertit vers le format de travail natif du SPU (32bits signés – *Fig. 16*).

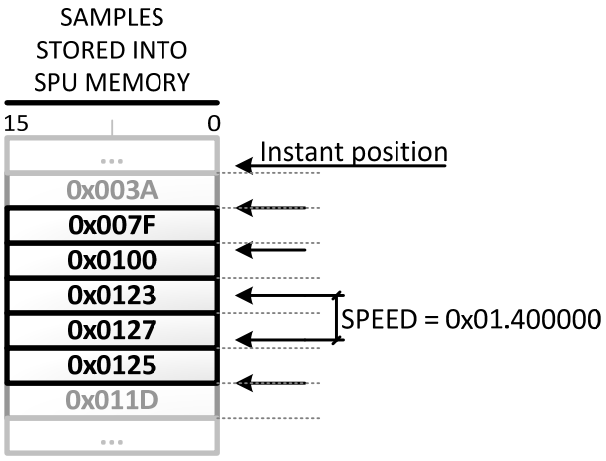


Fig. 15 – Concept : Vitesse de lecture des échantillons en mémoire RAM

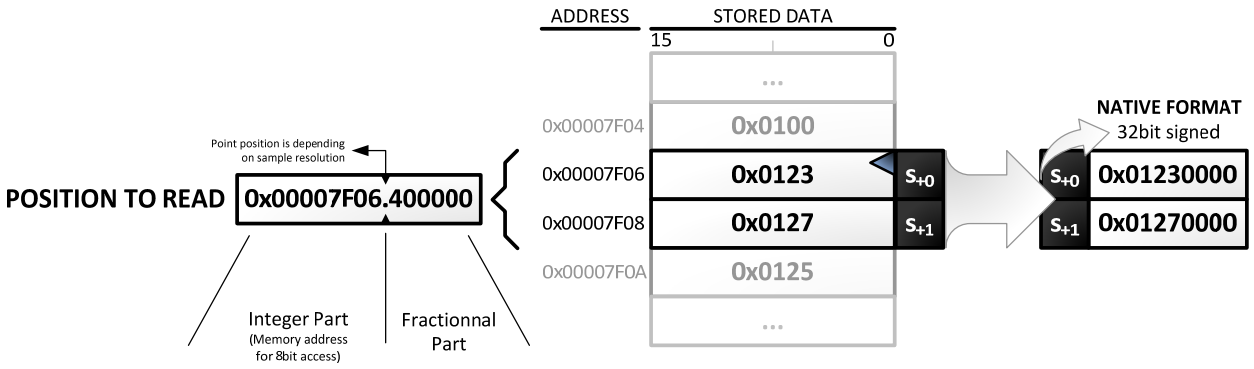


Fig. 16 – Concept : Collection et conversion des échantillons

L'unité « Interpolator » interpole les échantillons en entrée (*Fig. 17*), puis l'unité « Amplitude Modifier » atténue l'échantillon de sortie avant de l'envoyer finalement dans l'accumulateur.

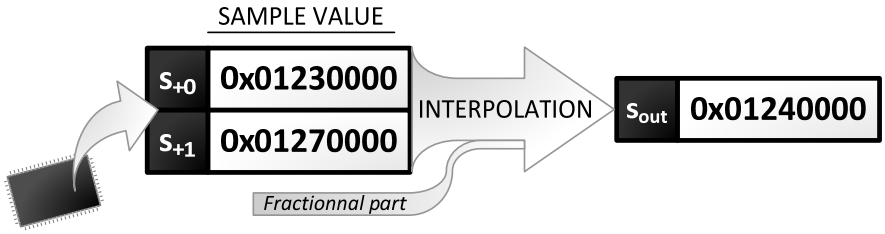


Fig. 17 – Concept : Interpolation

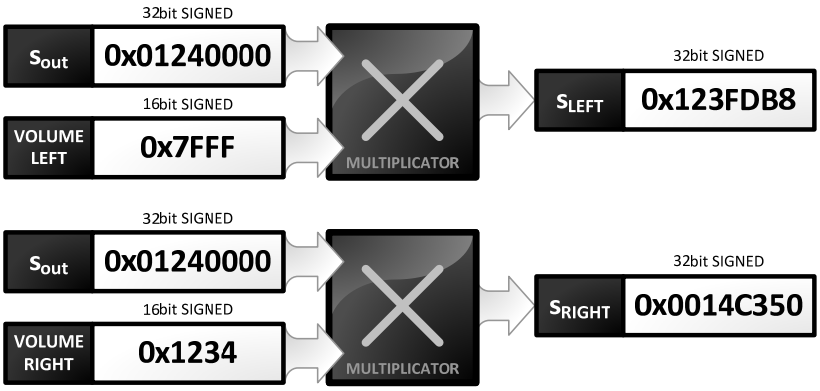


Fig. 18 – Concept : Modification de l'amplitude des échantillons

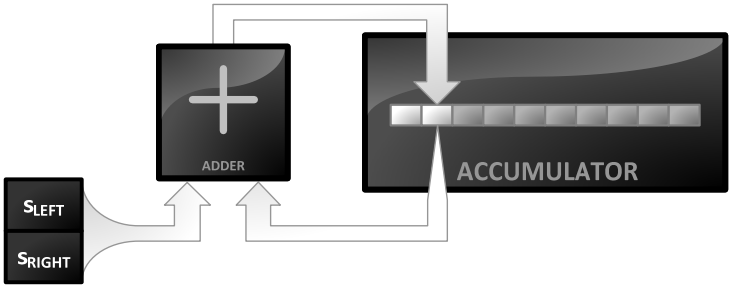


Fig. 19 – Concept : Accumulateur d'échantillons pour la formation du paquet

Les points suivants, complétés par l'annexe *«SCH0 »*, décrivent en détail le fonctionnement de chaque unité.

3.2.2.1 PRINCIPE DE COMMUNICATION ENTRE LES UNITÉS

Dans le canal physique de rendu sonore, les unités sont chaînées : à l'arrivée d'une nouvelle donnée, l'unité concernée calcule le résultat et le passe à la suivante. Pour faire en sorte que l'on puisse lancer le rendu des échantillons à la chaîne, avec un débit maximal et sans préoccupation de l'avancement des données dans chaque unité, le bus de communication suivant a été étudié :

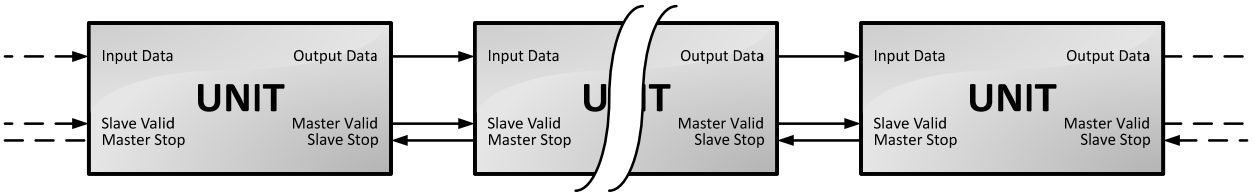


Fig. 20 – Principe de chaînage des unités

Voici la description des signaux de contrôle de flux d'une unité :

- **Slave Valid**
  Lorsque ce signal est à '1', les données présentes en entrée ("Input Data") sont valides au présent cycle d'horloge.
- **Slave Stop**
  Une des unités succédant à la courante unité demande à ce que toutes les unités précédentes cessent de produire des données.
- **Master Stop**
  La présente unité demande aux précédentes unités de ne plus lui fournir de données, car elle ne pourra pas les accepter.
- **Master Valid**
  La présente unité donne son résultat à l'unité suivante. Elle devra le maintenir tant que son propre "Slave Stop" sera à l'état haut.

Le diagramme suivant présente un exemple de comportement pour une unité dont le temps de traitement d'une donnée est de 1 cycle horloge, respectivement, 3 cycles horloge (non pipelinable) :

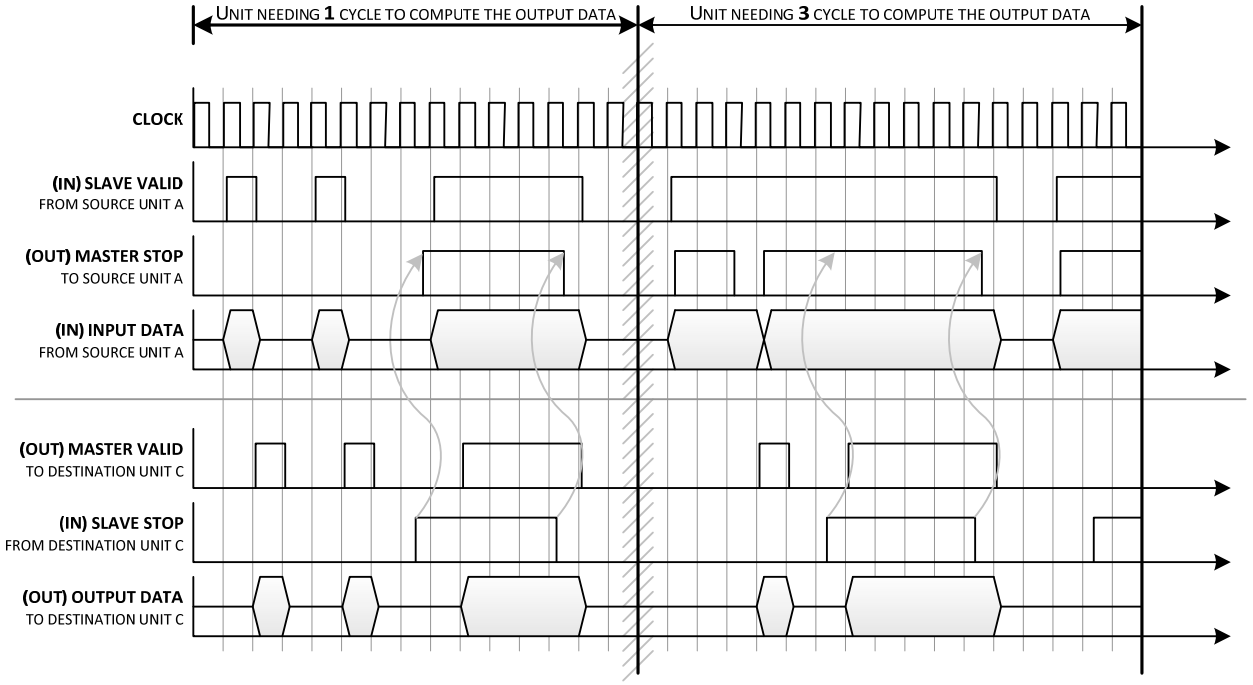


Fig. 21 – Diagramme de timing de la communication entre les unités

Les signaux « Valid » sont entièrement synchrones. Quant aux signaux « Stop », ils peuvent être soit :

- Asynchrones
- Synchrones (Avec '1' comme état par défaut - comportement similaire à un « Acknowledge ». Souffre néanmoins d'un débit maximum divisé par deux.)

3.2.2.2 UNITÉ « POSITION COMPUTER »

Cette unité a comme objectif de calculer la nouvelle position instantanée dans le fichier audio chargé en RAM en fonction de la vitesse de lecture qui lui est donnée.

Les fichiers sont caractérisés par plusieurs paramètres :

- Une position de départ absolue dans la mémoire
- Une première boucle de maintien (*sustain loop*)
- Une seconde boucle par défaut / de relâchement (default/*release loop*)

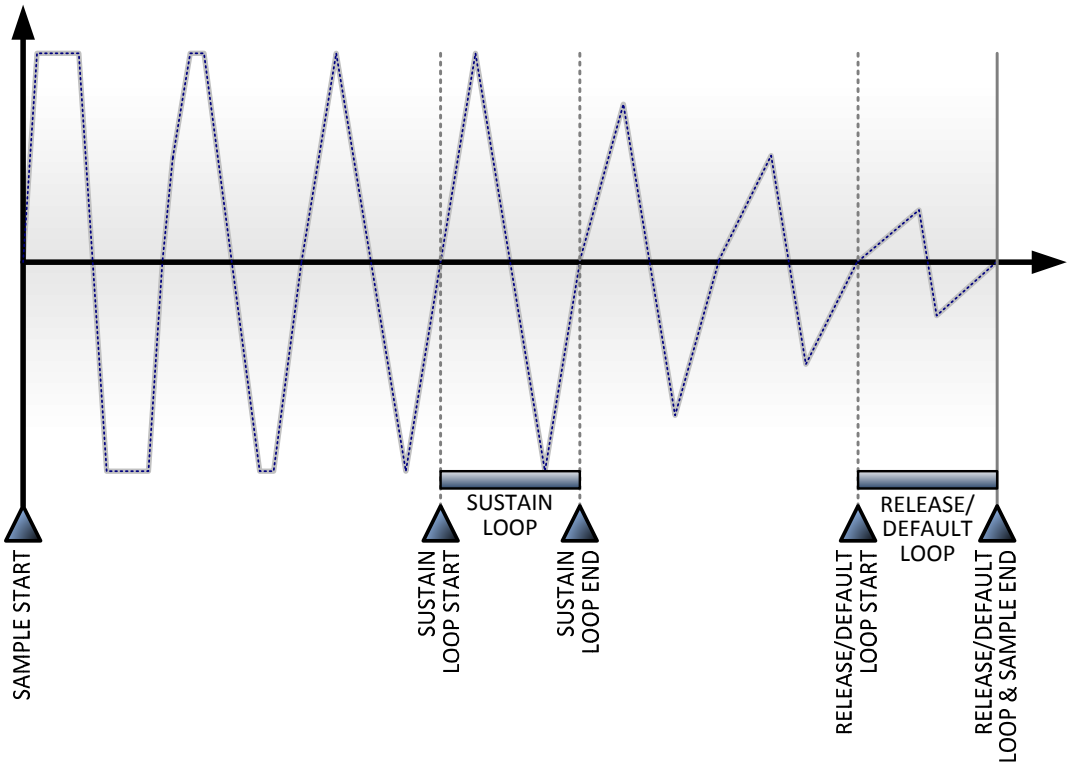


Fig. 22 – Positions indexées dans un fichier audio

Puisque qu'il est inutile de conserver les échantillons qui pourraient éventuellement suivre après la boucle de relâchement, la fin de celle-ci fait foi de fin du fichier.

Selon les besoins de l'utilisateur, ces boucles peuvent être activées/désactivées. Le tableau suivant montre leur intérêt :

Boucle de maintien	Boucle de relâchement / par défaut	Exemple d'utilisation
Désactivé	Désactivé	Utilisé pour les sons de courte durée et/ou sans possibilité de rebouclement (strictement non périodique). Concerne les bruitages et certains sons percussifs.
Désactivé	Activé	Utilisé pour les sons avec une attaque particulière (p.ex. : percussion de la corde d'un piano), mais suivis d'un comportement périodique. Concerne la grande majorité des instruments classiques (piano, basse,…) et synthétisés (strings, pads, leads…).
Activé	Désactivé	Cette combinaison peut trouver facilement usage avec des instruments issus de synthèse. Peut être également intéressant si l'on désire avoir une réverbération appliquée sur un instrument, sans toutefois disposer d'un réverbérateur à la sortie du SPU. Ainsi, l'instrument est enregistré avec sa réverbération. Utilisable dans le cas où la boucle de maintien est de très faible durée et le relâchement rapide, suivi de la réverbération (p.ex. : piano ou vibraphone dans une salle de concert). Doit être utilisé avec une modulation en amplitude adéquate.
Activé	Activé	Optimisation du point précédent par le placement d'un extrait de la réverbération dans la seconde boucle. Doit être utilisé avec une modulation en amplitude adéquate. Permet également d'obtenir des effets intéressants avec des instruments issus de synthèse (il s'agit de l'utilisation la plus probable).

Tabl. 2 – Présentation des boucles de maintien (sustain) et de relâchement(release) / de défaut

Dans les boucles, la lecture des échantillons du fichier peut être monodirectionnelle ou bidirectionnelle (lorsque la fin de la boucle est atteinte, le fichier est lu à l'envers jusqu'au début de la boucle où il repartira dans son sens normal). L'illustration suivante (*Fig. 23*) présente bien ces comportements :

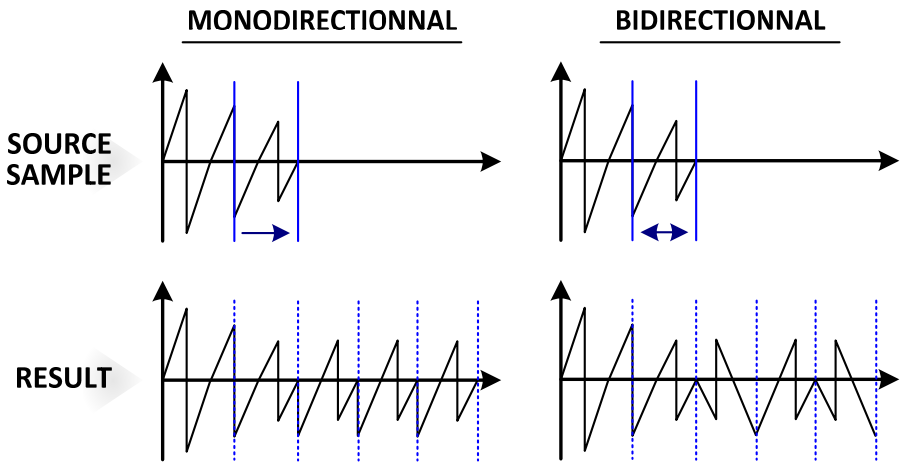


Fig. 23 – Boucle monodirectionnelle & bidirectionnelle

3.2.2.2.1 Réalisation numérique

La position dans le fichier peut être caractérisée à n'importe quel instant par un état spécifique :

- **In delay**
  Lorsqu'un canal a été initialisé et que son traitement est commencé par l'unité, il est possible que l'utilisateur ait spécifié un délai avant la lecture des échantillons. Ainsi, tant que le compteur de délai ne sera pas nul, le premier échantillon du fichier sera maintenu en sortie.

- **In Sustain Loop**
  L'unité se considère dans cet état si la boucle de maintien est active (note maintenue) et que la position actuelle de lecture se situe dans cette boucle. Si active, la boucle de maintien est prioritaire sur celle de relâchement.
- **In Release Loop**
  L'unité se considère dans cet état si la boucle de relâchement/défaut est active et que la position actuelle de lecture se situe dans cette boucle.
- **Go ahead**
  C'est l'état par défaut : ni dans une boucle ni dans le délai de départ. Si la fin du fichier est atteinte, le dernier échantillon est maintenu en sortie.

NB : A cause du maintien du premier et du dernier échantillon dans les états « In delay », respectivement, « Go ahead », il est recommandé que les fichiers audio utilisés en mémoire commencent et se terminent sur une valeur nulle (si aucune boucle de relâchement/défaut n'est présente).

Le calcul de la nouvelle position se fait en deux étapes (deux cycles non pipelinable) : la première consiste à déterminer la nouvelle position et ce, indépendemment de tout paramètre. En parallèle est déterminé l'état dans lequel on se situe (avant la nouvelle position). Dans la seconde étape, en fonction de l'état déterminé précédemment, la nouvelle position finale est calculée. A partir de ce résultat sont évaluées deux cases mémoire : la première étant la partie entière de la position et la seconde étant la première incrémentée de 1 (le comportement diffère lorsque qu'on se situe au seuil d'une boucle – voir annexe « SCH0 – Page 1»).

3.2.2.3 UNITÉ « SAMPLE COLLECTOR »

Le « Sample Collector » fait avant tout office de bridge d'accès à la mémoire RAM : ainsi, si le SPU venait à être porté sur une autre board avec un bus d'accès mémoire différent (p.ex. : strictement 8bits ou 32bits par opposition à la généricité du multiplexeur d'accès réalisé qui supporte 8bits, 16bits et 32bits), seul ce bloc devrait être modifié.

Il assure également que les échantillons lus soient convertis vers le format de travail natif du SPU, soit 32bits signé. L'unique format accepté en entrée est le LPCM (Linear Pulse Code Modulation) sous les formes suivantes :

- **Sign** **:** *Unsigned, Signed*
- **Endianness** **:** *Little, Big*
- **Quantification** **:** *8bit, 16bit, 32bit*

Les échantillons stockés sur 24bits ne sont pas supportés pour des raisons de non alignement avec les cases 32bits de la mémoire de travail. Cela entraînerait en effet une dégradation des performances puisque deux accès mémoire seraient nécessaires pour reconstituer une partie des échantillons. Pour contourner cette restriction, ceux-ci doivent être copiés dans la mémoire de travail en tant qu'échantillons 32bits.

Cette unité prend également en compte le type d'interpolation qui devra être appliquée par la suite. Ainsi, seuls les accès nécessaires seront effectués (voir unité « Interpolator », au point suivant).

3.2.2.4 UNITÉ « INTERPOLATOR »

Cette unité permet d'interpoler les échantillons fournis par le « Sample Collector ». Cette opération est nécessaire, puisque la vitesse de lecture des échantillons est en général une valeur à virgule, ce qui nécessite de lire des positions situées 'entre deux cases mémoires'.

Deux types d'interpolation sont supportés :

- **Sample & Hold**
  Aucune interpolation – l'échantillon +0 (soit la partie entière de la position) est directement transmis au bloc suivant :

  $$s_{OUT} = s_0$$

  Avec : s_0 = Echantillon à l'adresse correspondant à la partie entière de la position.

- **Linear**
  L'interpolation linéaire consomme les échantillons +0 et +1 selon la formule suivante :

  $$s_{OUT} = s_0 + (s_1 - s_0) \cdot fractionalPart$$

  Avec : s_0 = Echantillon à l'adresse correspondant à la partie entière de la position.
  s_1 = Echantillon à l'adresse $s_0 + 1$
  $fractionalPart$ = Partie fractionnaire de la position ($0 \leq fractionalPart < 1$)

3.2.2.5 UNITÉ « AMPLITUDE MODIFIER »

L'unité « Amplitude Modifier » a pour but de modifier l'amplitude de l'échantillon interpolé selon les constantes de volume gauche et droit du canal logique selon le comportement suivant (les volumes gauche et droit étant signés, l'inversion de phase est possible) :

$$s_R = s_{OUT} \cdot \frac{volume_{RIGHT}}{2^{volume_{RIGHT}'length-1}}$$

$$s_L = s_{OUT} \cdot \frac{volume_{LEFT}}{2^{volume_{LEFT}'length-1}}$$

NB : Il est fortement recommandé que les fichiers audio préchargés par l'utilisateur ne présentent pas d'offset continu : cela risquerait de provoquer des parasites, voire des débordements en sortie.

3.2.2.6 UNITÉ « ACCUMULATOR »

L'unité « Accumulator » réceptionne les échantillons calculés pour chaque canal logique dans une petite mémoire SRAM de taille fixe définissant la taille maximale d'un paquet. Chaque échantillon rendu du canal logique '*N*' sera additionné en mémoire à la position correspondant au numéro d'échantillon dans le paquet en cours.

L'unité reçoit trois signaux de contrôle venant du « Manager », soit :

- Un signal permettant de se repositionner au début de la mémoire pour débuter la réception des échantillons d'un nouveau canal logique rendu.
- Un signal pour vider le paquet complet dans la FIFO du convertisseur D/A. A la suite de cette opération, les échantillons du prochain canal logique rendu seront directement copiés dans la mémoire, écrasant ainsi le paquet précédent. Pour les canaux logiques suivants, les échantillons seront à nouveau additionnés au contenu de la mémoire.

- Un signal pour envoyer '*m*' (soit la taille du paquet) échantillons de valeur nulle en sortie. Ce mécanisme est important pour conserver la présence du flux audio sur le D/A en cas d'absence totale de canaux logiques actifs.

Elle offre aussi un signal au « Manager » indiquant sa position dans la mémoire de stockage, ce qui lui permet de savoir si tous les échantillons ont été rendus. Dans ce cas, le switching context de canal peut avoir lieu. Elle informe également le « Manager » lorsqu'une opération de vidage d'un paquet sur le D/A est en cours.

Cette unité est directement câblée sur la FIFO (tampon) du D/A.

NB : L'unité n'est pas protégée contre les débordements de capacité. Dans un tel cas, ce n'est pas un phénomène de distorsion qui apparaîtrait, mais bien de sévères parasites. La responsabilité de ne pas faire saturer la sortie est laissée à l'utilisateur.

3.2.3 ACCÈS AUX REGISTRES ET À LA MÉMOIRE DE TRAVAIL DU SPU

Les registres de configuration et de contrôle du SPU, ainsi que tous les registres des canaux logiques sont centralisés dans le bloc « Registers and Memory Access Manager » (voir *Fig. 11*). D'un côté, celui-ci implémente une interface directement mappée sur le CPU, offrant l'accès à tous les registres et à la mémoire de travail du SPU. De l'autre côté, il donne l'accès aux registres de configuration/contrôle au « Manager » et s'occupe de placer le bon canal logique dans le canal physique sur commande du « Manager ».

3.2.3.1 INTERFACE AVALON

La communication entre le CPU et le SPU a lieu par une interface Avalon de type Memory-Mapped (Avalon-MM) dont voici le comportement, issu des spécifications[14] :

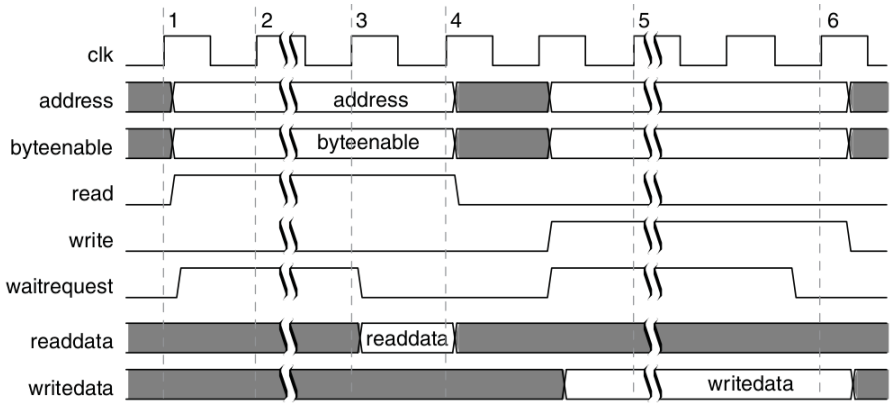


Fig. 24 – Diagramme de timing du bus Avalon-MM

NB : L'accès au SPU est strictement 32bits. Ainsi, le signal « byteenable » n'est pas présent. Néanmoins, bien que l'adressage soit de type 8bits, le CPU ne devrait jamais accéder à une adresse non alignée sur 32bits (les deux bits du LSB non à '0').

[14] Infos complémentaires : *http://www.altera.com/literature/manual/mnl_avalon_spec.pdf*, page 3-8

3.2.3.2 REGISTRES DE CONFIGURATION/CONTRÔLE

Voici les principaux registres de configuration et de contrôle du SPU :

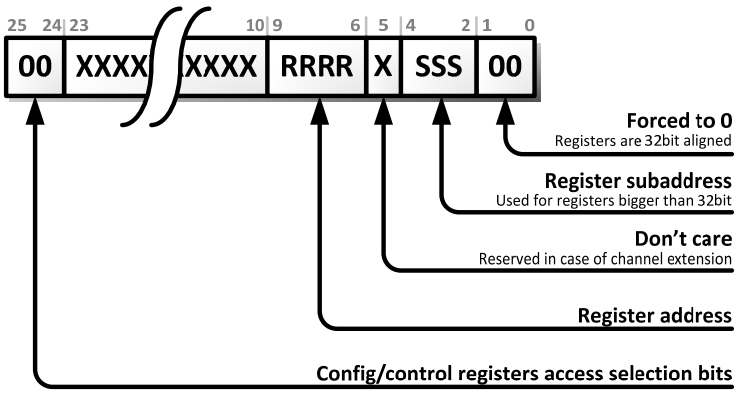


Fig. 25 – Formatage de l'accès aux registres de configuration/contrôle du SPU par le CPU

Offset	Taille	Nom	Accès CPU	Accès SPU	Description
CONTRÔLE DIRECT					
RRRR : 0_h SSS : Don't care	1bit	FREEZE	RW	R	Mise en pause du SPU à la fin du rendu du paquet actuel.
RRRR : 1_h SSS : Don't care	8bits	PACKET RENDER SIZE	RW	R	Taille « N-1 » en échantillons d'un paquet audio (c'est-à-dire le nombre d'échantillons rendus pour chaque canal avant un changement de contexte. (P.ex. : 0x0F → paquets de 16 échantillons)
RRRR : 2_h SSS : 0_h to 7_h	256 x 1bit	CHANNEL EN	RW	R	Contrôle direct de l'activation des canaux logiques.
RRRR : 3_h SSS : 0_h to 7_h	256 x 1bit	CHANNEL SUSTAIN EN	RW	R	Contrôle direct du maintien (*sustain*) des canaux logiques.
MUTEX POUR LA MODIFICATION DES CANAUX LOGIQUES					
RRRR : 8_h SSS : Don't care	8bits	CHANNEL SEL	W	R	Sélection du canal.
RRRR : 9_h SSS : Don't care	1bit	CHANNEL WRLOCK	W	R	Demande d'accès exclusif aux registres du canal.
RRRR : A_h SSS : Don't care	1bit	CHANNEL LOCKED	R	W	Accès exclusif au canal spécifié autorisé (le canal sera verrouillé en accès au SPU pendant que CHANNEL WRLOCK et CHANNEL LOCKED seront à '1')
DEBUG / MESURE DES PERFORMANCES▲					
RRRR : C_h SSS : Don't care	32bits	PACKET COMPUTE TIME	R	W	Temps (en cycle d'horloge) pour le calcul d'un paquet et envoi sur le D/A. Permet de déceler une surcharge du SPU.
RRRR : D_h SSS : Don't care	32bits	MEMORY WRITE ACCESS PER SECOND	R	W	Nombre d'écritures par seconde sur la mémoire de travail du SPU.
RRRR : E_h SSS : Don't care	32bits	MEMORY READ ACCESS PERS ECOND	R	W	Nombre de lectures par seconde sur la mémoire de travail du SPU.
RRRR : F_h SSS : Don't care	1bit	CRITICAL ERROR	R	W	Flag de debug mis à '1' en cas d'erreur critique. Concerne le débordement des FIFOs du multiplexeur d'accès à la mémoire de travail, ainsi que de celles du contrôleur SSRAM – cas (ne devant jamais apparaître) qui mettrait le système dans un état strictement indéterminé.

▲ Ces registres pourront servir dans le futur à la régulation de l'usage du SPU par le driver. Ils permettraient notamment de déterminer le nombre de canaux logiques maximal avec la plus faible latence de rendu possible.

Tabl. 3 – Liste des registres configuration/contrôle du SPU

3.2.3.3 REGISTRES DES CANAUX LOGIQUES

Les registres descripteurs des canaux logiques sont stockés dans une SRAM interne à la FPGA et sont accessibles par le CPU de la manière suivante :

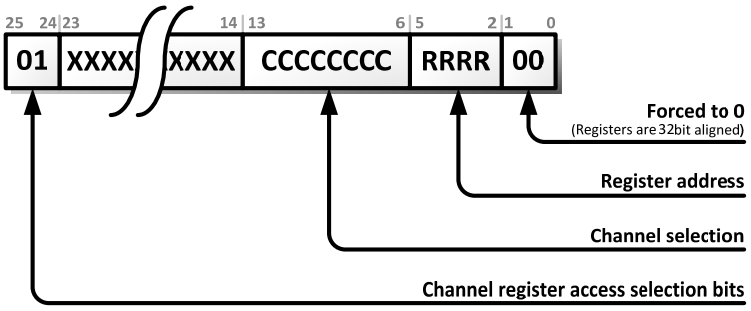


Fig. 26 – Formatage de l'accès aux registres des canaux logiques par le CPU

Offset	Bits		Nom	Description
RRRR : 0_h	1..0	✗	Sample Size	Taille des échantillons : ▪ 00_b : 8bits ▪ 10_b : 32bits ▪ 01_b : 16bits ▪ 11_b : 8bits
	2	✗	Sample Endianess	Endianness des échantillons : ▪ 0_b : Little endian ▪ 1_b : Big endian
	3	✗	Sample Sign	Signe des échantillons : ▪ 0_b : Unsigned ▪ 1_b : Signed
	4	✓	Interpolation Type	Type d'interpolation : ▪ 0_b : Sample & Hold ▪ 1_b : Linear
	5..7		Don't care	
	8	✗	Sustain loop enable	Etat de la boucle de sustain : ▪ 0_b : OFF ▪ 1_b : ON
	9	✗	Sustain loop type	Comportement de la boucle de sustain : ▪ 0_b : Monodirectionnal ▪ 1_b : Bidirectionnal
	10	✗	Release/default loop enable	Etat de la boucle de release/défaut : ▪ 0_b : OFF ▪ 1_b : ON
	11	✗	Release/default loop type	Comportement de la boucle de release/défaut : ▪ 0_b : Monodirectionnal ▪ 1_b : Bidirectionnal
	12..31		Don't care	.
RRRR : 1_h	31..0	✗	Sample start	Position absolue du début du fichier audio (premier échantillon) dans la mémoire de travail du SPU.
RRRR : 2_h	31..0	✗	Sustain loop start	Position absolue de l'échantillon délimitant le début de la boucle de maintien (*sustain*) du fichier audio.
RRRR : 3_h	31..0	✗	Sustain loop end	Position absolue de l'échantillon délimitant la fin de la boucle de maintien (*sustain*) du fichier audio (est compris dans la boucle).
RRRR : 4_h	31..0	✗	Release/Default loop start	Position absolue de l'échantillon délimitant le début de la boucle de release/défaut du fichier audio.
RRRR : 5_h	31..0	✗	Sample end	Position absolue de la fin du fichier audio (dernier échantillon) dans la mémoire de travail du SPU.
RRRR : 6_h	15..0	✓	Right volume	Volume gauche (signé – inversion de phase possible).
	31..16	✓	Left volume	Volume droit (signé – inversion de phase possible)
RRRR : 7_h	31..0	✓	Speed	Vitesse de lecture (référencée à la fréquence de mixage du SPU).
RRRR : 8_h	23..0	✗	Delay▲	Délai avant le début de la lecture des échantillons (si différent de 0, maintien le premier échantillon en sortie). Décrémenté à chaque cycle d'horloge.
	30..24		Don't care	
	31	✗	Read direction▲	Sens de la lecture des échantillons : ▪ 0_b : Forward (Direct) ▪ 1_b : Backward (Inverse)
RRRR : 9_h	7..0		Don't care	
	31..8	✗	Position – fractionnal part▲	Partie fractionnaire (virgule) de la position actuelle dans la mémoire de travail. Devrait être initialisé à 0.
RRRR : A_h	31..0	✗	Position – integer part▲	Partie entière de la position actuelle dans la mémoire de travail. Devrait être initialisé à la même valeur que « Sample start ».
RRRR : B_h to F_h	31..0		Don't care	Réservé pour des modifications/améliorations futures.

Légende :

- ✘ Ces registres ne devraient pas être édités une fois le canal logique paramétré pour une lecture (cela n'aurait pas grand sens…sauf pour des cas très spécifiques et contrôlés).
- ✓ Ces registres peuvent sans autre être édités à n'importe quel moment.
- ▲ Il s'agit de registres de travail des unités SPU qui doivent être initialisés à chaque (re)paramétrage intégral du canal logique. Cependant, ils ne doivent plus être modifiés par la suite.

Tabl. 4 – Liste des registres des canaux logiques

3.2.3.4 ACCÈS À LA MÉMOIRE DE TRAVAIL DU SPU

Une interface du multiplexeur d'accès à la mémoire de travail du SPU est directement accessible par le CPU à la manière d'une mémoire classique, en lecture et en écriture :

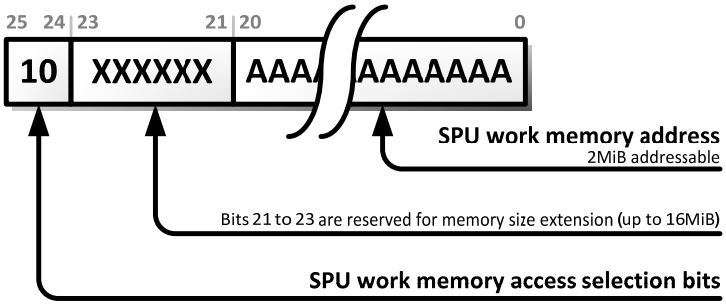


Fig. 27 – Formatage de l'accès à la mémoire de travail du SPU par le CPU

3.2.4 MANAGER

Le « Manager » du SPU a pour mission de séquencer le rendu des canaux logiques, c'est-à-dire de placer successivement les canaux logiques dans le canal physique pour effectuer le rendu des paquets. La figure suivante illustre ce principe :

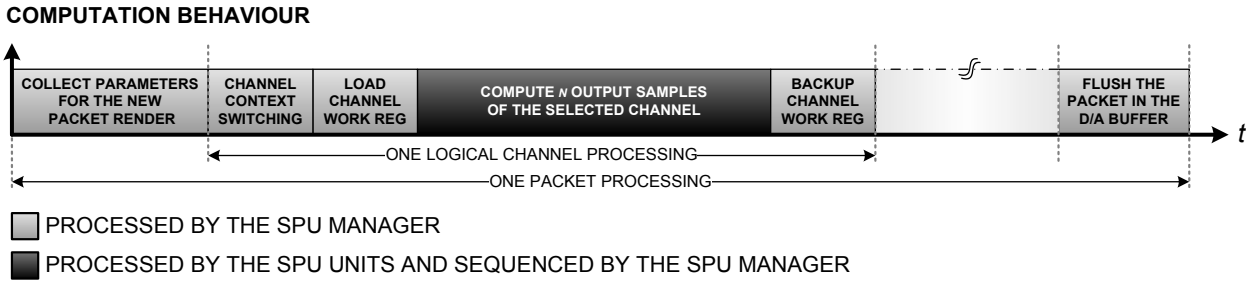


Fig. 28 – Représentation graphique du rendu d'un paquet

Le schéma détaillé de cette unité se trouve sous l'annexe « SCH2 ».

3.2.4.1 EXCLUSION MUTUELLE

Un problème important pourrait apparaître si le CPU venait à modifier à la volée les canaux logiques actifs (bit concerné du "CHANNEL EN" à '1' - voir *Tabl. 3*) :

Fig. 29 – Illustration du problème de l'interaction simultanée du CPU et du SPU sur un canal logique

En effet, si le canal logique était chargé dans le canal physique au moment où le CPU a réécrit ses registres, toute modification (y compris sur les registres constants) serait perdue lorsque le « Manager » donnerait l'ordre de sauvegarder les registres du canal physique vers le logique.

Pour ne pas se retrouver dans ce cas, le CPU doit impérativement utiliser les registres de configuration/contrôle "CHANNEL SEL", " CHANNEL WRLOCK" et "CHANNEL LOCKED" sur les canaux logiques actifs de la manière suivante :

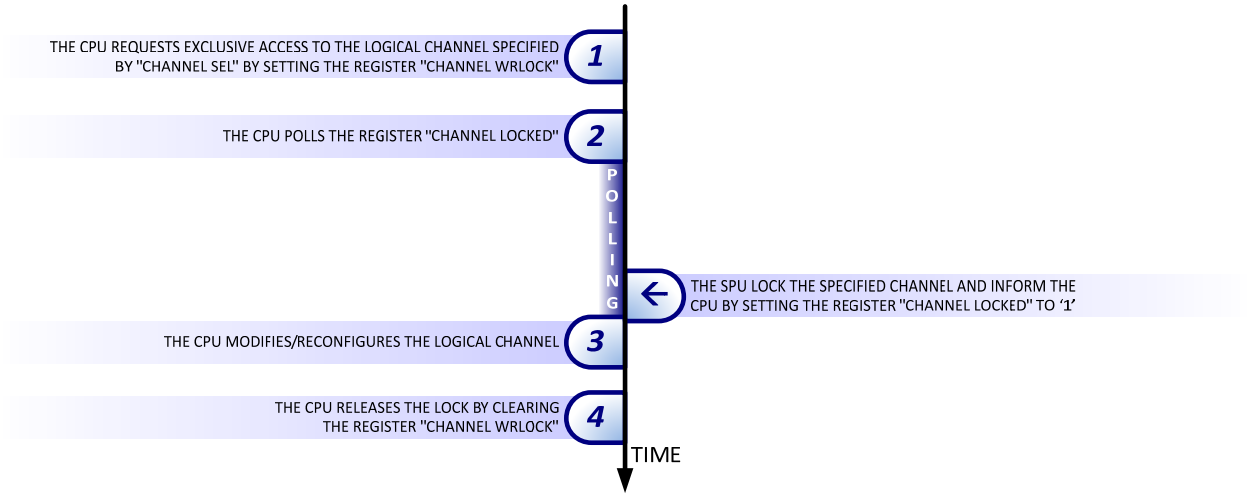


Fig. 30 – Procédure de modification d'un canal actif par le CPU

3.2.5 A PROPOS DU CONVERTISSEUR D/A

Le SPU est directement interfacé sur le port d'un bloc pilotant le D/A de la board, implémenté dans le passé et repris "tel quel" pour le projet. Ce port n'est rien d'autre qu'une FIFO dans laquelle le SPU jette les échantillons rendus.

Une fois le système mis en service, ce bloc s'occupe de paramétrer automatiquement le D/A via I^2C pour un fonctionnement à 88.2kHz - 24bits - stéréo. Il génère également un simple VU-mètre pour l'utilisateur et offre la possibilité de régler et de couper le volume de sortie. Voici une illustration des interactions possibles avec le D/A via ce bloc :

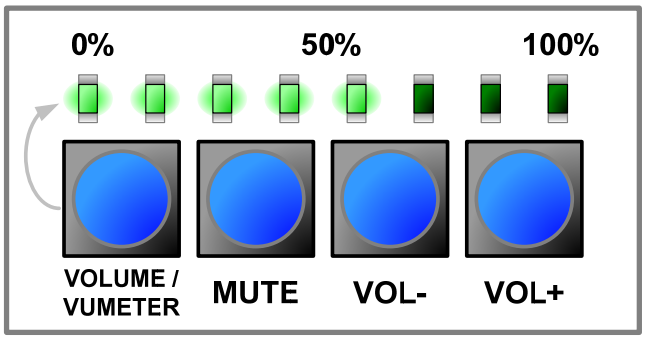


Fig. 31 – Interface de contrôle du codec audio

3.3 PARTIE CPU

3.3.1 CONFIGURATION HARDWARE

Le CPU est basé sur un softcore Nios*II*. Voici sa mise en œuvre dans 'QSys', l'outil d'intégration système de Quartus*II* :

Use	Name	Description	Clock	Base	End	IRQ
☑	clk50M	Clock Source				
☑	clk125M	Clock Source				
☑	mem_sdramCtrl	SDRAM Controller	clk125M	0x00000000	0x03ffffff	
☑	mem_mmClockCrossingBridge	Avalon-MM Clock Crossing Bridge	*multiple*	**0x00000000**	0x03ffffff	
☑	cpu_Nios2	Nios II Processor	clk50M	0x06000000	0x060007ff	
☑	externBus_HSPU	External Access Port	clk50M	**0x10000000**	0x1fffffff	
☑	cpuP_mmBridge	Avalon-MM Pipeline Bridge	clk50M	**0x07000000**	0x0700ffff	
☑	cpuP_SystemID	System ID Peripheral	clk50M	0x00000000	0x00000007	
☑	cpuP_timer0	Interval Timer	clk50M	0x00000100	0x0000011f	0
☑	cpuP_jtagUart	JTAG UART	clk50M	0x00000200	0x00000207	1
☑	cpuP_uart0	UART (RS-232 Serial Port)	clk50M	0x00000400	0x0000041f	2
☐	cpuP_uart1	UART (RS-232 Serial Port)	*unconnected*			
☑	cpuP_pio7seg_x4	PIO (Parallel I/O)	clk50M	0x00000800	0x0000080f	
☑	cpuP_pio7seg_x2_0	PIO (Parallel I/O)	clk50M	0x00000810	0x0000081f	
☑	cpuP_pio7seg_x2_1	PIO (Parallel I/O)	clk50M	0x00000820	0x0000082f	
☑	cpuP_pioSwitches_x18	PIO (Parallel I/O)	clk50M	0x00000830	0x0000083f	
☑	cpuP_pioLEDs_x18	PIO (Parallel I/O)	clk50M	0x00000840	0x0000085f	
☑	cpuP_epcsFlashCtrl	EPCS Serial Flash Controller	clk50M	0x00004000	0x000047ff	3

Fig. 32 – Structure générale du CPU

Le processeur Nios*II* (*cpu_Nios2)*, cadencé à 50MHz, est relié à deux bridges et un bus :

- **Bridge CPU⇆SDRAM (*mem_mmClockCrossingBridge*)**
  La SDRAM étant cadencée à 125MHz et le CPU à 50MHz, ce bridge synchronise les signaux pour chaque domaine d'horloge respectif.

Use	Connections	Name	Description	Clock	Base	End	IRQ
☑		mem_sdramCtrl	SDRAM Controller				
	→	clk	Clock Input	clk125M			
	→	reset	Reset Input	[clk]			
	→	s1	Avalon Memory Mapped Slave	[clk]	0x00000000	0x03ffffff	
	○	wire	Conduit				
☑		mem_mmClockCrossingBridge	Avalon-MM Clock Crossing Bridge				
	→	m0_clk	Clock Input	clk125M			
	→	m0_reset	Reset Input	[m0_clk]			
	→	s0_clk	Clock Input	clk50M			
	→	s0_reset	Reset Input	[s0_clk]			
	→	s0	Avalon Memory Mapped Slave	[s0_clk]	**0x00000000**	0x03ffffff	
	←	m0	Avalon Memory Mapped Master	[m0_clk]			
☑		cpu_Nios2	Nios II Processor				
	→	clk	Clock Input	clk50M			
	→	reset_n	Reset Input	[clk]			
	←	data_master	Avalon Memory Mapped Master	[clk]			
	←	instruction_master	Avalon Memory Mapped Master	[clk]			
	→	d_irq	Interrupt Receiver	[clk]	IRQ 0	IRQ 31	
	×←	jtag_debug_module_reset	Reset Output	[clk]			
	→	jtag_debug_module	Avalon Memory Mapped Slave	[clk]	0x06000000	0x060007ff	
	×←	custom_instruction_master	Custom Instruction Master				

Fig. 33 – Connexion CPU⇆SDRAM via un bridge (adaptation du domaine horloge)

- **Bus CPU⇆SPU (*externBus_HSPU*)**
  Le SPU est directement mappé sur le CPU en tant que slave Avalon-MM.

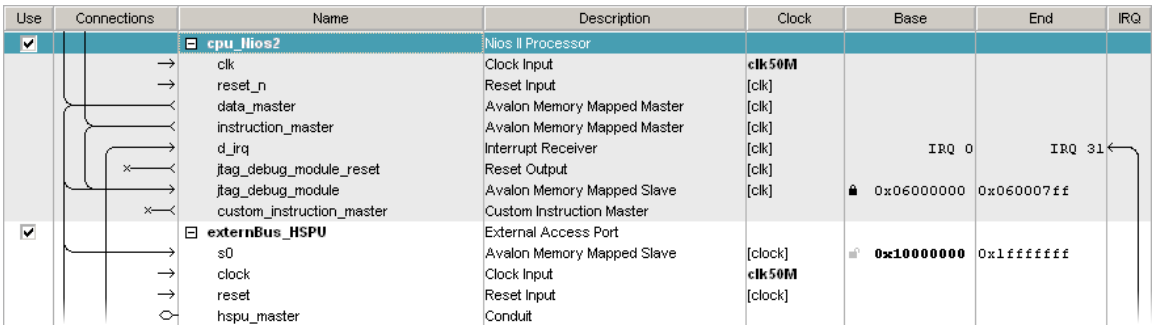


Fig. 34 – Connexion CPU⇆SPU via un bus Avalon

- **Bridge CPU⇆Périphériques (*cpuP_mmBridge*)**
  Sont reliés sur le master de ce bridge tous les périphériques du CPU. Le clock des périphériques est le même que celui du CPU, soit 50MHz.

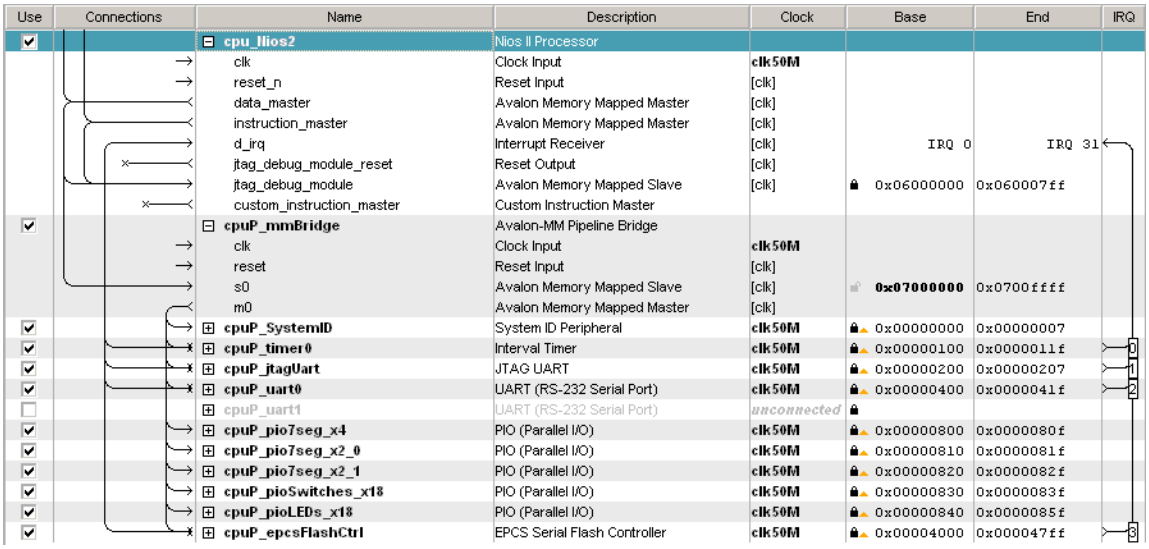


Fig. 35 – Connexion CPU⇆Périphériques via un bridge

Les périphériques du CPU sont :

- **SystemID**
  Retourne un numéro d'identification du système qui change à chaque synthétisation du système. Evite l'exécution d'anciens codes n'étant plus adaptés au système.
- **Timer0**
  Un simple timer pour le CPU.
- **JTAG UART**
  Interface UART par JTAG. Utile pour du débogage.
- **UART**
  Interface UART pour le MIDI.
- **PIO pour les 8x 7segments, 18x switches, 18x LEDs**
  Pour du contrôle et du débogage.
- **Contrôleur Flash EPCS**
  Contrôleur d'accès à la mémoire FLASH de configuration de la FPGA. Cette mémoire possède une certaine quantité d'espace libre utilisable pour le code du CPU et pour le stockage des banques sonores. Actuellement inutilisé.

Voici le mapping mémoire qui a été défini :

	cpu_Nios2.data_master	cpu_Nios2.instruction_master	mem_mmClockCrossingBridge.m0	cpuP_mmBridge.m0
mem_mmClockCrossingBridge.s0	0x00000000 - 0x03ffffff	0x00000000 - 0x03ffffff		
mem_sdramCtrl.s1 via mem_mmClockCr...	0x00000000 - 0x03ffffff	0x00000000 - 0x03ffffff		
cpu_Nios2.jtag_debug_module	0x06000000 - 0x060007ff	0x06000000 - 0x060007ff		
cpuP_mmBridge.s0	0x07000000 - 0x0700ffff			
cpuP_SystemID.control_slave via cpuP...	0x07000000 - 0x07000007			
cpuP_timer0.s1 via cpuP_mmBridge	0x07000100 - 0x0700011f			
cpuP_jtagUart.avalon_jtag_slave via cp...	0x07000200 - 0x07000207			
cpuP_uart0.s1 via cpuP_mmBridge	0x07000400 - 0x0700041f			
cpuP_pio7seg_x4.s1 via cpuP_mmBridge	0x07000800 - 0x0700080f			
cpuP_pio7seg_x2_0.s1 via cpuP_mmBr...	0x07000810 - 0x0700081f			
cpuP_pio7seg_x2_1.s1 via cpuP_mmBr...	0x07000820 - 0x0700082f			
cpuP_pioSwitches_x18.s1 via cpuP_m...	0x07000830 - 0x0700083f			
cpuP_pioLEDs_x18.s1 via cpuP_mmBri...	0x07000840 - 0x0700085f			
cpuP_epcsFlashCtrl.epcs_control_port ...	0x07004000 - 0x070047ff			
externBus_HSPU.s0	0x10000000 - 0x1fffffff			
cpuP_SystemID.control_slave				0x00000000 - 0x00000007
cpuP_timer0.s1				0x00000100 - 0x0000011f
cpuP_jtagUart.avalon_jtag_slave				0x00000200 - 0x00000207
cpuP_uart0.s1				0x00000400 - 0x0000041f
cpuP_pio7seg_x4.s1				0x00000800 - 0x0000080f
cpuP_pio7seg_x2_0.s1				0x00000810 - 0x0000081f
cpuP_pio7seg_x2_1.s1				0x00000820 - 0x0000082f
cpuP_pioSwitches_x18.s1				0x00000830 - 0x0000083f
cpuP_pioLEDs_x18.s1				0x00000840 - 0x0000085f
cpuP_epcsFlashCtrl.epcs_control_port				0x00004000 - 0x000047ff
mem_sdramCtrl.s1			0x00000000 - 0x03ffffff	

Fig. 36 – Mapping mémoire

NB : Actuellement, le CPU n'a qu'une seule interface UART, ce qui limite le contrôle du système à un seul périphérique MIDI. Celle-ci est par ailleurs routée sur le seul port connecteur SUB-D9 de la carte de développement, avec un baudrate fixé à 38'400 Baud/s pour être utilisé avec le driver Yamaha CBX-MIDI[15]. Celui-ci offre la possibilité d'utiliser le port RS232 du PC comme port MIDI. Cette solution permet d'éviter temporairement de réaliser un adaptateur MIDI DIN-5 (côté périphérique) vers SUB-D9 (côté FPGA).

Pour pouvoir utiliser une connectique DIN-5, il est nécessaire de changer le baudrate à 31'250 Baud/s (fréquence non supportée par les contrôleurs RS232 des PCs).

[15] Page de téléchargement : *http://download.yamaha.com/cbx_midi/index.html*
(Ne supporte pas les OS Microsoft plus récents que Windows NT5.X)

3.3.2 RÉALISATION SOFTWARE

3.3.2.1 PROGRAMME DE TEST #1 – SYNTHÉTISEUR SOFT « SAWTOOTH »

Pour s'assurer du bon fonctionnement du CPU et de sa périphérie, un code de test a été réalisé avant l'implémentation du SPU. Il s'agit d'un synthétiseur software très basique réceptif aux messages MIDI de notes (avec leur volume) et jouant des dents de scie :

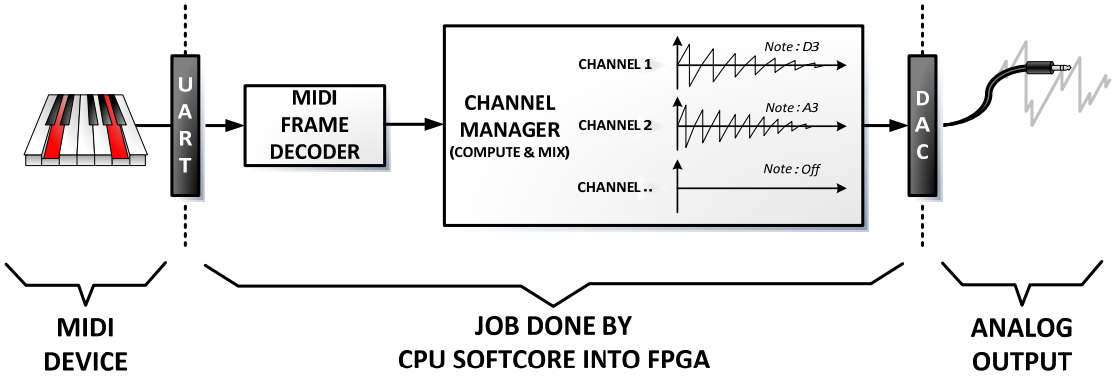


Fig. 37 – Synthétiseur sawtooth : Comportement

Le diagramme de classe de la Fig. 39 présente sa structure.

« Manager » instancie une interface MIDI « MidiInterface » qui décode et traite les événements MIDI reçus sur un UART spécifié lors de sa création (la réception a lieu par interruption hardware). Il instancie également *'N'* canaux « SawtoothChannel » pour le rendu du son. Son travail est de lire les messages MIDI reçus et d'initialiser les canaux avec les paramètres correspondants. « SawtoothChannel » est étudié pour générer des dents de scie au comportement suivant :

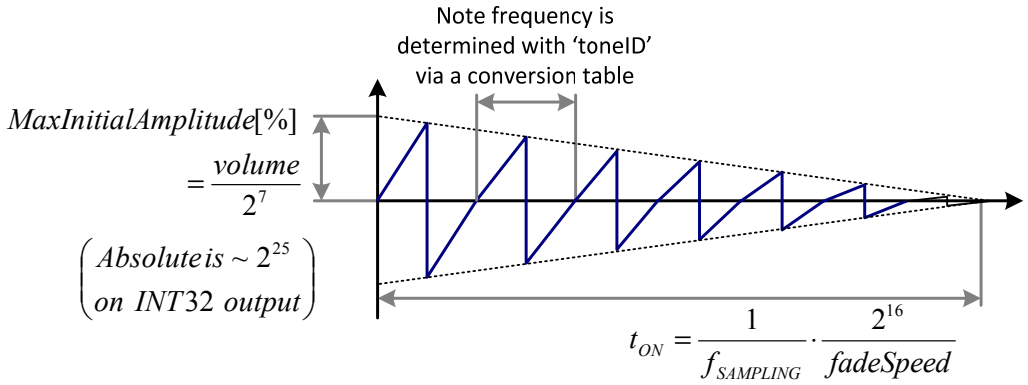


Fig. 38 – Synthétiseur sawtooth : Comportement du « SawtoothChannel »

Le code source en C++ de cette application de test se trouve sur le CD-ROM (projet Nios*II* «SOFTSPU»).

NB : Bien que les sources soient existantes dans la liste de projets du workspace d'Eclipse, cette application de test ne peut plus être exécutée sur la version finale du HSPU (la périphérie du Nios*II* ayant changé entre temps).

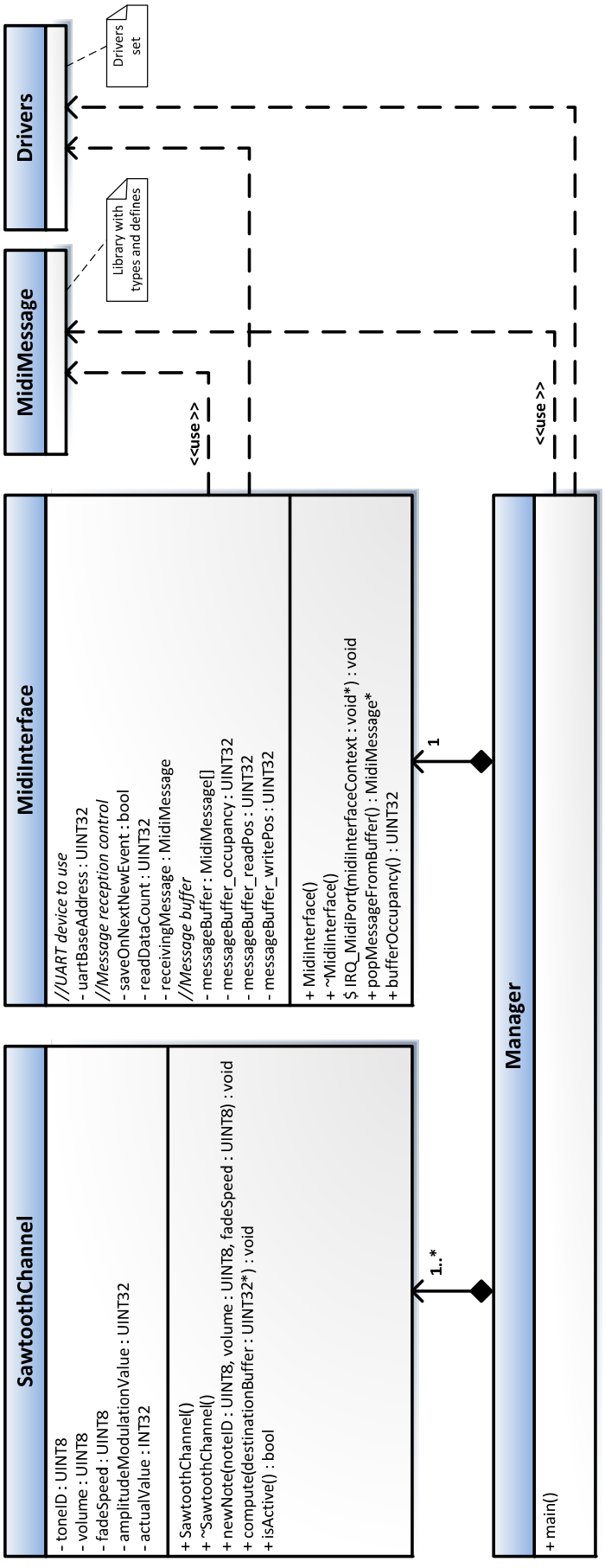


Fig. 39 – Diagramme de classe UML du synthétiseur sawtooth

3.3.2.2 PROGRAMME DE TEST #2 – HARDWARE « SQUARE GENERATOR »

Pour tester le bon fonctionnement du SPU, a été étudié un code de test C++ pilotant le SPU pour le faire générer un carré à partir de la superposition de sinus. Celui-ci peut être trouvé sur le CD-ROM (projet Nios*II* « SQUAREGEN_TESTER »).

Ce code de test génère un carré à partir d'un sinus chargé au démarrage dans le SPU. L'utilisateur peut régler le nombre d'harmoniques à l'aide des switches SW6 à SW0 (max. 128 harmoniques).

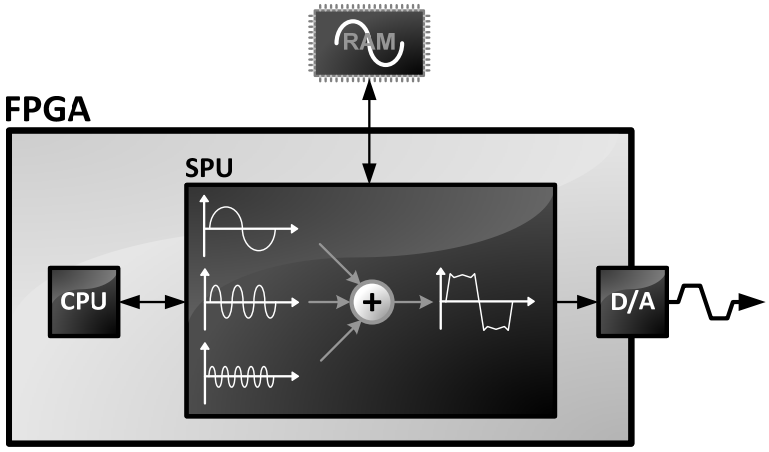


Fig. 40 – « Square Generator » : Comportement

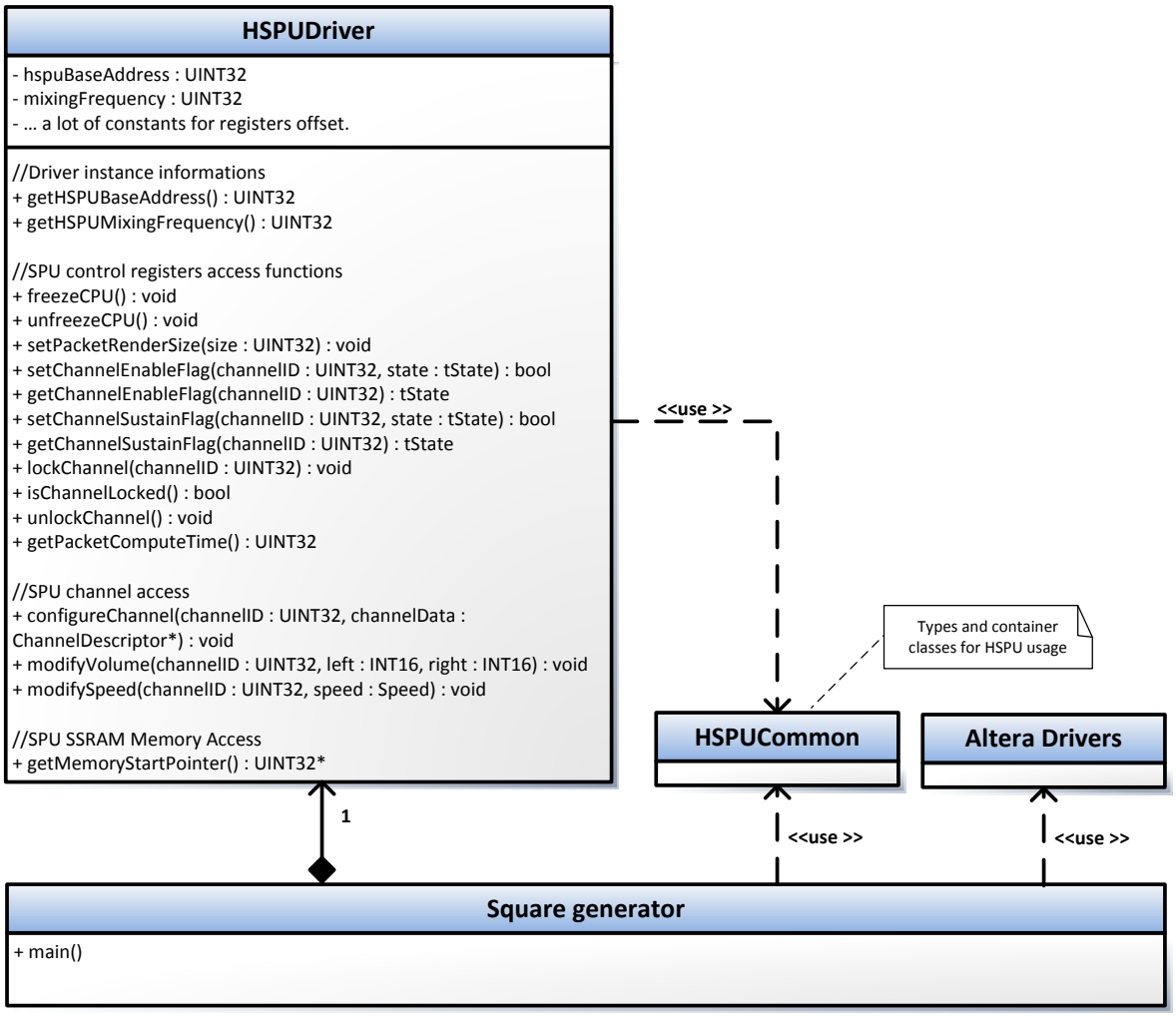


Fig. 41 – Diagramme de classe UML du « Square generator »

3.3.2.3 SYNTHÉTISEUR MIDI « SQUAREHARP »

Pour démontrer le bon fonctionnement et la possibilité d'exploiter le HSPU à des fins musicales, un synthétiseur MIDI mono-instrumental a été implémenté. Cet instrument a été réalisé selon les critères suivants :

- Sollicitation du maximum de canaux logiques simultanément.
- Exploitation de plusieurs fichiers audio, remplissant la quasi totalité de la mémoire de travail du SPU.
- Réceptif à tous les événements MIDI concernant le jeu de nouvelle notes, avec considération de leur amplitude. Les événements en provenance du canal MIDI 10 (soit celui réservé aux percussions) sont ignorés.

Le code source est disponible sur le CD-ROM (projet Nios*II* « MIDISYNTH »).

3.3.2.3.1 Réalisation de l'instrument

L'instrument de démonstration, intitulé « SquareHarp » est basé sur une onde (presque) carrée et a été réalisé avec le logiciel de synthèse ZynAddSubFX dont voici une capture avec l'instrument en question :

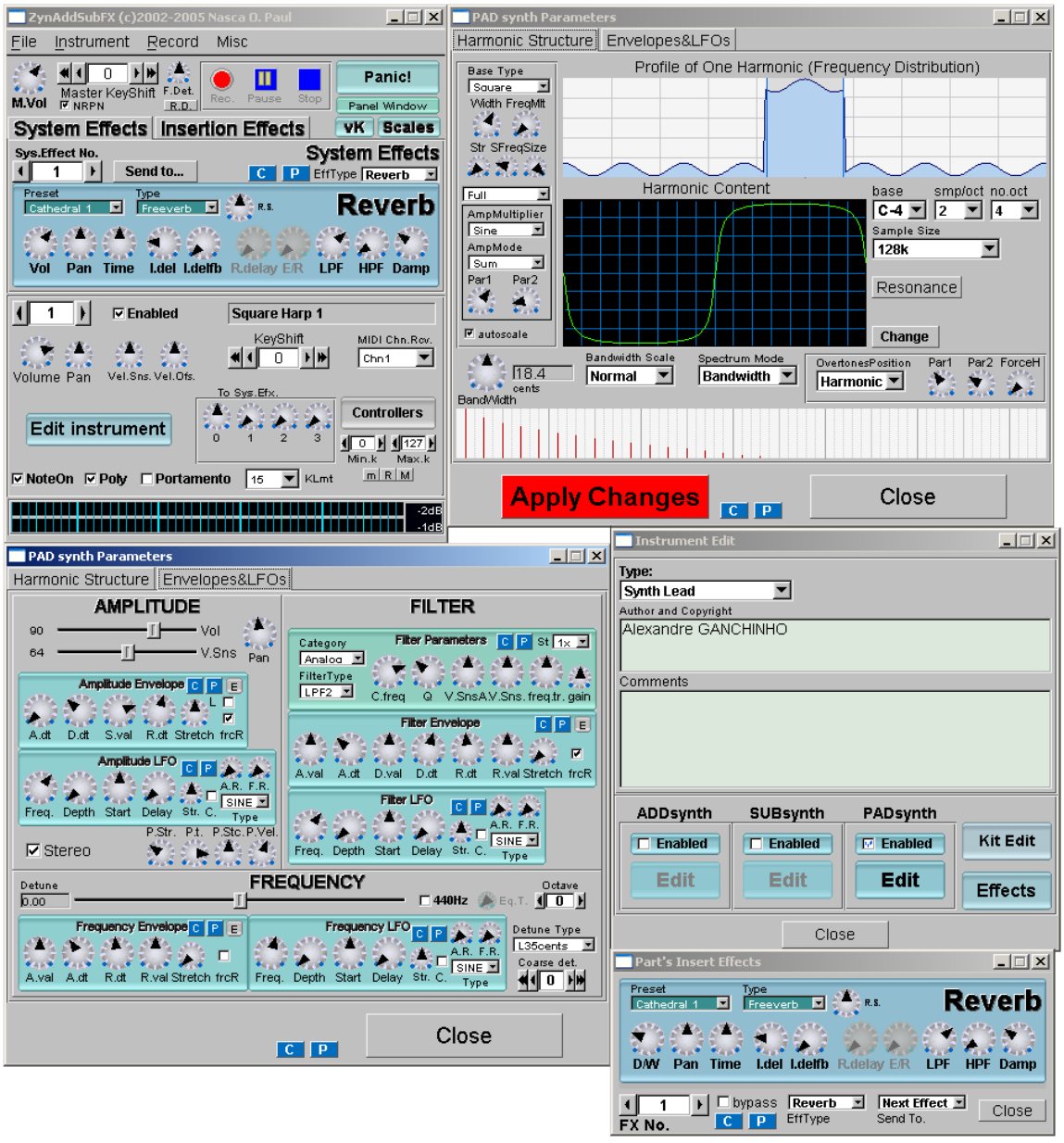


Fig. 42 – Instrument « SquareHarp » sous le logiciel de synthèse sonore ZynAddSubFX

Son comportement est très percussif, « pulsé », avec une longue réverbération. Il a été échantillonné en 44.1kHz 16bits stéréo sur 4 notes (pour une taille totale de 2MB) dont voici le mapping :

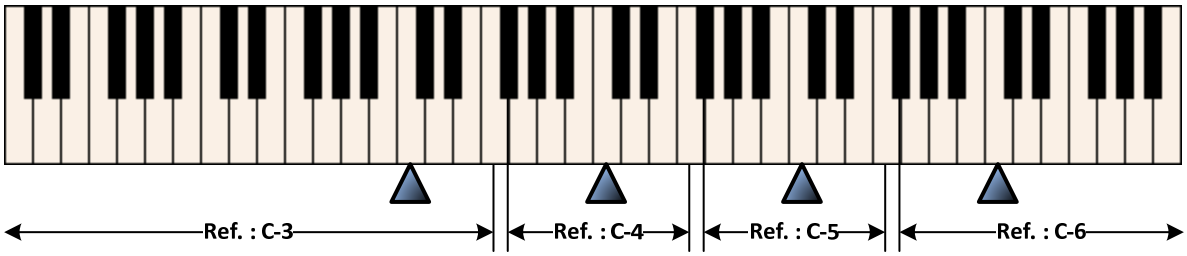


Fig. 43 – Mappage des notes échantillonnées de l'instrument « SquareHarp »

Pour les notes en-dessous de la moitié de l'octave 2, la vitesse de lecture de la note échantillonnée en C-3 est suffisamment lente pour que son attaque rapide laisse place à une basse particulièrement « ronde » et « profonde » (la réverbération dure bien plus longtemps). Quant aux notes supérieures, elles ont un comportent staccato (« piquant ») avec un son très clair.

3.3.2.3.2 Réalisation du code C++

Le code de ce synthétiseur réceptionne les événements MIDI et exploite le SPU via les drivers implémentés précédemment dans les deux codes de tests (voir *Fig. 45*).

Le comportement général du code est exprimé dans le diagramme de flux ci-dessous :

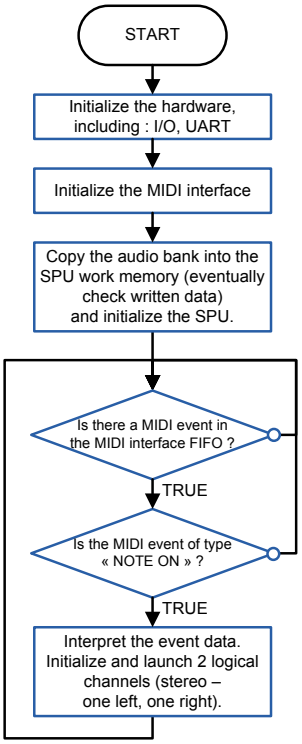


Fig. 44 – Diagramme de flux du code de l'instrument « SquareHarp »

Bien que cela aurait été un grand plus pour ce projet, aucune mémoire FLASH n'a été utilisée pour le stockage des échantillons (auquel cas il aurait aussi fallu réaliser un système de descripteur d'instruments de sorte à avoir une banque sonore). Les échantillons sont directement présents sous forme de tableaux dans des fichiers header *(.h)*. Au vu de la taille du fichier à charger (2MB), a été réalisé un petit utilitaire convertissant n'importe quel fichier vers un tableau en C.

A l'initialisation du système, les échantillons, présents dans la mémoire SDRAM du CPU, sont copiés dans la mémoire de travail du SPU. Une fois cela fait, le code effectue en boucle les opérations de collecte des événements MIDI, leur interprétation et la configuration des canaux logiques.

Pour les notes à jouer reçues, deux modes d'allocation des voix existent :

- **(Semi-)Monophonique** *(1 MIDI Channel[16] – 1 voice)*
  Chaque canal MIDI est assigné à une voix (une paire de canaux logiques pour l'enregistrement de gauche et de droite). Chaque nouvelle note sur un même canal MIDI coupe net la précédente. Essentiellement destiné à des fins de déboguage.
- **Polyphonique** *(Round robin)*
  Tous les canaux logiques du SPU sont exploités tour à tour. Cet instrument se prêtant bien à ce comportement, cela évite toute coupure de notes et fait ressortir pleinement sa

[16] Pour rappel, la norme MIDI assigne chaque événement (note, contrôle) à un canal MIDI (sur un total de 16). Les événements globaux ne sont toutefois pas concernés.

réverbération. Dans le cas du jeu rapide d'un très grand nombre de notes simultanées, seules les notes les plus anciennes seront coupées.

Le choix d'un de ces deux modes passe par le switch *SW0*.

Le nombre de canaux logiques maximum à utiliser, ainsi que la taille de rendu des paquets est configurable par quelques *defines*. Par défaut, ces valeurs sont de 192 canaux (96 voix), respectivement, 64 échantillons.

Il est à noter qu'un léger effet de panoramique (*panning*) est appliqué aléatoirement sur chaque note jouée.

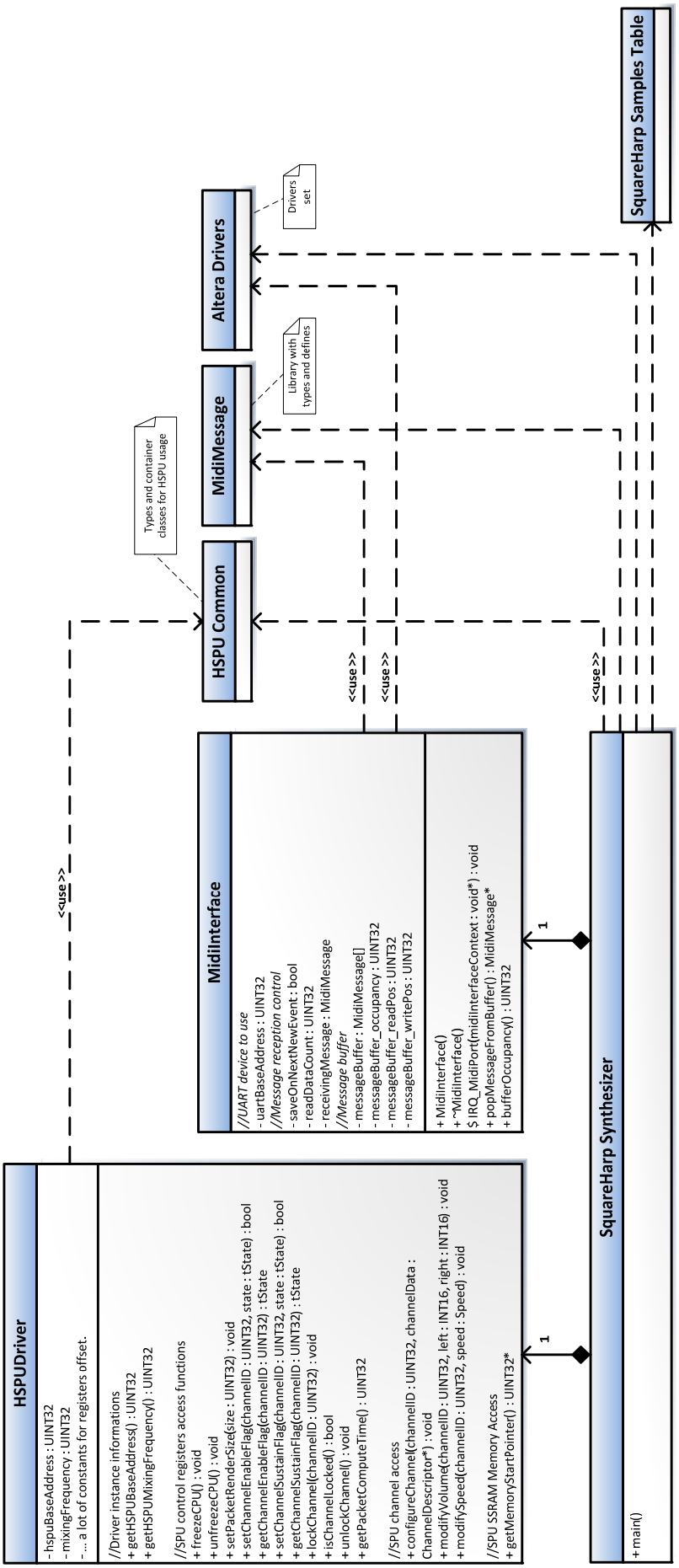


Fig. 45 – Diagramme de classe UML du synthétiseur MIDI «SquareHarp»

4. VERIFICATION

Afin de s'assurer du bon fonctionnement du projet, certains tests ciblés, puis globaux ont été effectués. Ceux-ci sont décrits dans cette section.

4.1 TESTS UNITAIRES DU SPU

Chaque bloc VHDL codé a été minutieusement testé avec succès dans sa fonctionnalité sous ModelSim. Voici quelques extraits de simulation des blocs les plus « visuels ».

4.1.1 UNITÉS DE TRAITEMENT SONORE DU CANAL PHYSIQUE

La première figure (*Fig. 46*) correspond à la représentation graphique de la position absolue dans la lecture des échantillons, calculée par l'unité « Position Computer ». L'unité y subit d'ailleurs trois changements de contexte. Les premières positions calculées (de la gauche jusqu'à la première cassure net vers le centre) sont celles d'un canal logique configuré avec une boucle de *sustain* monodirectionnelle et une boucle de *release/default* bidirectionnelle. Le second canal chargé (au centre) est chargé avec une boucle de *sustain* bidirectionnelle et aucune boucle de *release*. Le troisième (à droite) n'a qu'une boucle de *release/default*. Sa vitesse est accélérée dans le temps.

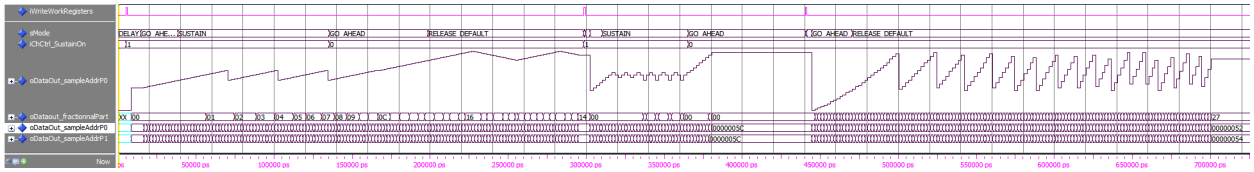


Fig. 46 – Extrait de simulation sous ModelSim de l'unité « Position Computer »

La figure suivante (*Fig. 47*) présente l'interpolation (unité « Interpolator ») entre deux échantillons aux valeurs opposées :

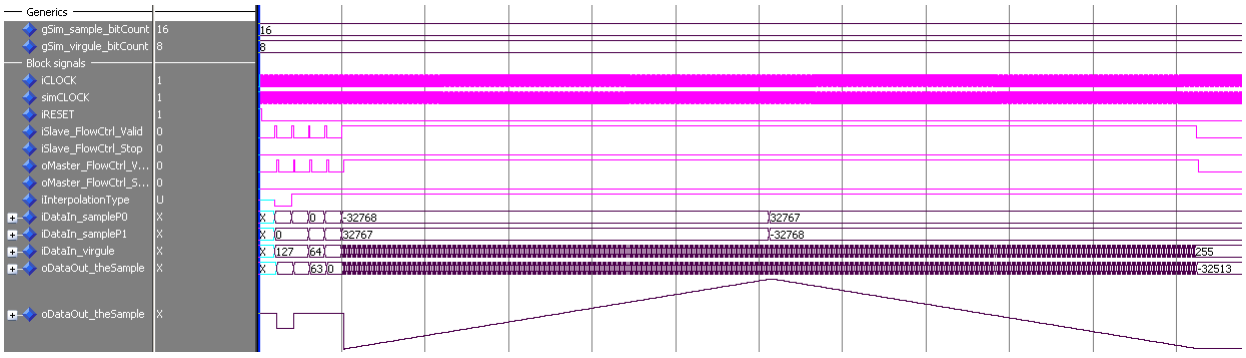


Fig. 47 – Extrait de simulation sous ModelSim de l'unité « Interpolator »

Finalement, la figure ci-dessous (*Fig. 48*) présente l'unité « Amplitude Modifier », atténuant diverses valeurs constantes en entrée par une rampe de volume couvrant toute la plage :

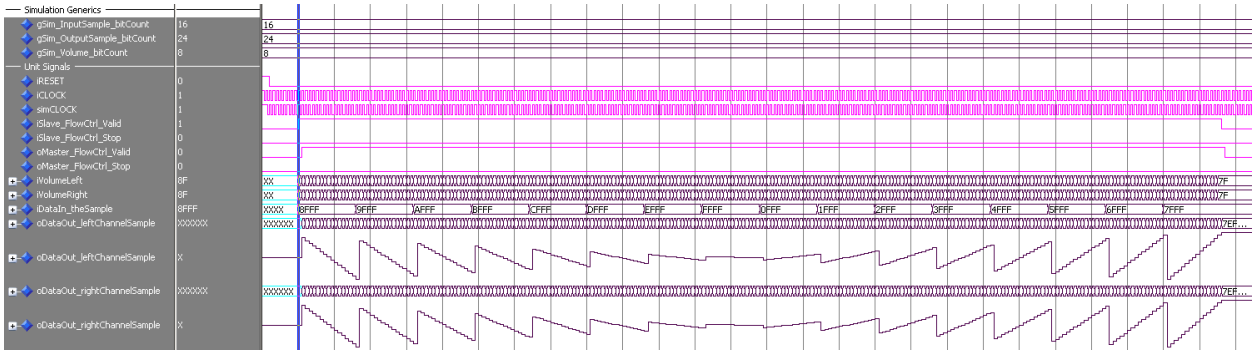


Fig. 48 – Extrait de simulation sous ModelSim de l'unité « Amplitude Modifier »

4.1.2 CONTRÔLEUR SSRAM

Le contrôleur SSRAM a été plus délicat à la simulation qu'à l'implémentation… En effet, il existe en général des modèles de simulation (issus des fabricants) pour tester les contrôleurs. Malheureusement, le seul trouvé était en Verilog… or la version Altera de ModelSim est bridée contre le mixage VHDL et Verilog. Finalement, après quelques essais hardware infructueux, l'IP pour SSRAM de QSys (dont l'usage n'aurait pas convenu pour des raisons de dépendance) a été analysée et son comportement reproduit.

L'illustration ci-dessous présente l'activité générée par le contrôleur SSRAM sur celle-ci au moment du rendu d'un canal logique. Cette mesure a été effectuée avec l'outil SignalTap*II* Logic Analyser, intégré à Quartus*II*.

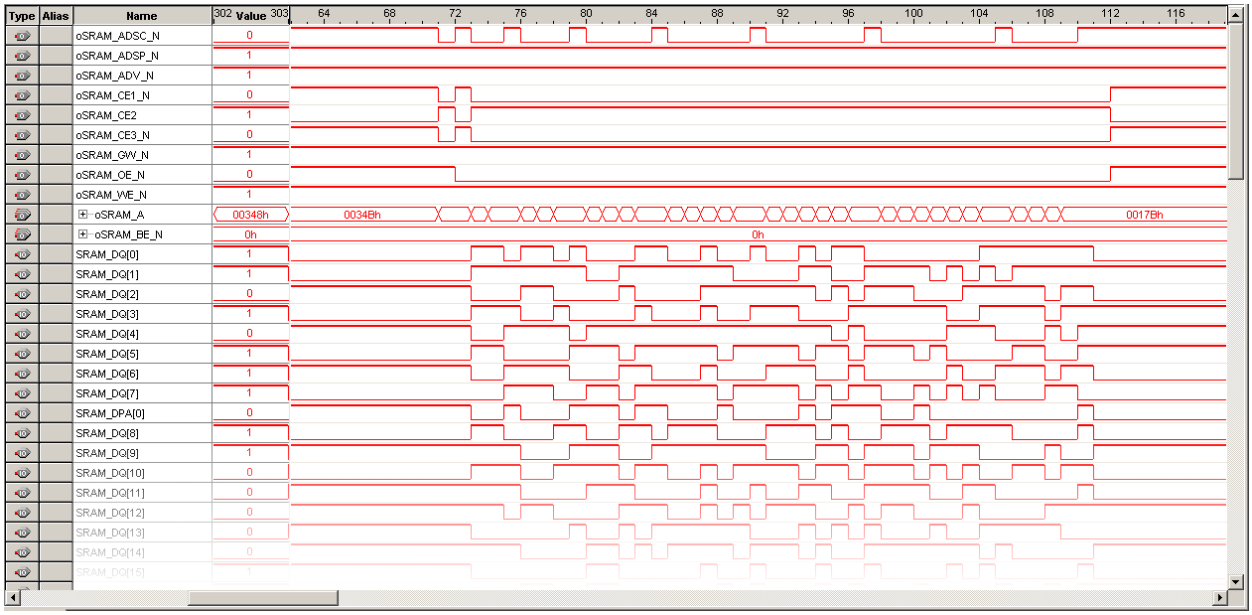


Fig. 49 – Analyse de l'activité sur la SSRAM

4.2 TESTS FONCTIONNELS DU SOFTCORE, DE L'INTERFACE MIDI ET DU CODEC AUDIO

Voici la liste des tests effectués avec le synthétiseur soft « Sawtooth » (voir chapitre 3.3.2.1) et leurs résultats :

ORDRE	DESCRIPTION DU TEST	SUCCÈS
#1	**Relier la board au PC par RS232 (connexion MIDI). Relier également la sortie audio sur l'entrée du PC en vue d'analyser le son. Programmer la FPGA et exécuter le code du CPU.** Le texte « HSPU » doit s'afficher sur les 7-segments.	✓
#2	**Ouvrir un fichier MIDI avec un quelconque séquenceur MIDI et le paramétrer pour qu'il joue sur la sortie RS232 (nécessite l'installation du driver YAMAHA CBX-MIDI). Lancer la lecture.** Les LEDs rouges indiquent le nombre d'événements MIDI reçus et joués. La board génère bien du son en sortie (son volume est réglable à l'aide des deux boutons bleus de droite) et l'on peut voir les LEDs vertes se comporter comme des vumètres.	✓
#3	**Observer l'allure du son avec le PC.** Le son doit clairement ressembler à des dents de scie.	✓

Tabl. 5 – Liste des tests effectués avec le synthétiseur soft Sawtooth

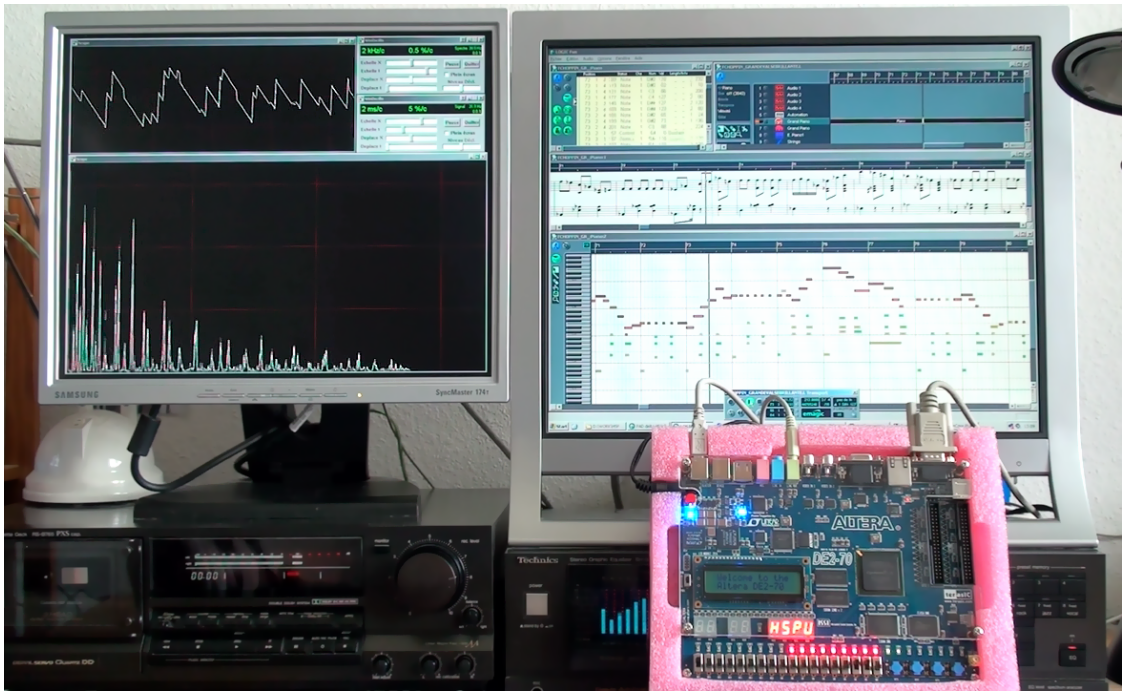


Fig. 50 – Jeu d'une mélodie polyphonique sur le synthétiseur soft Sawtooth[17]

La *Fig. 50* présente le synthétiseur Sawtooth jouant les notes que le PC lui transmet par MIDI. Sur l'écran de gauche, on peut voir la forme de l'onde et la FFT de la sortie du D/A de la carte de développement. Ces résultats positifs montrent que le CPU softcore Nios*II*, ainsi que toute sa périphérie (SDRAM, UART, I/O) sont fonctionnels.

Cependant, les performances maximales atteintes avec ce synthétiseur très basique ne sont que de 24 canaux « Sawtooth » au maximum pour un rendu mono à 22.5kHz, 32bits. Nous sommes donc bien loin des spécifications du SPU hardware (256 canaux, 88.2kHz, 32bits stéréo), et ce pour une fonctionnalité plus basique. Cela prouve que les capacités de rendu sonore en software sont très en-dessous de celles en hardware.

[17] Extrait de la vidéo de démonstration du synthétiseur Sawtooth, disponible sur le CD-ROM.

4.3 TESTS FONCTIONNELS DE BASE DU SPU

Voici quelques mesures réalisées avec le programme de test #2 « Square Generator » (voir chapitre 3.3.2.2). Elles ont été effectuées directement à la sortie du D/A à l'aide d'une carte son ($f_{SAMPLING} = 44.1kHz$,16bits) et du logiciel « Baudline[18] » :

Fig. 51 – Signal généré avec : Nbre d'échant. du sinus = 4 / Nbre de sinus = 1 / Fréq. = 100Hz

Ce sinus, composé de seulement 4 points, donne un triangle grâce à l'interpolation linéaire.

[18] Réf. : *http://www.baudline.com* - Logiciel Opensource pour Linux

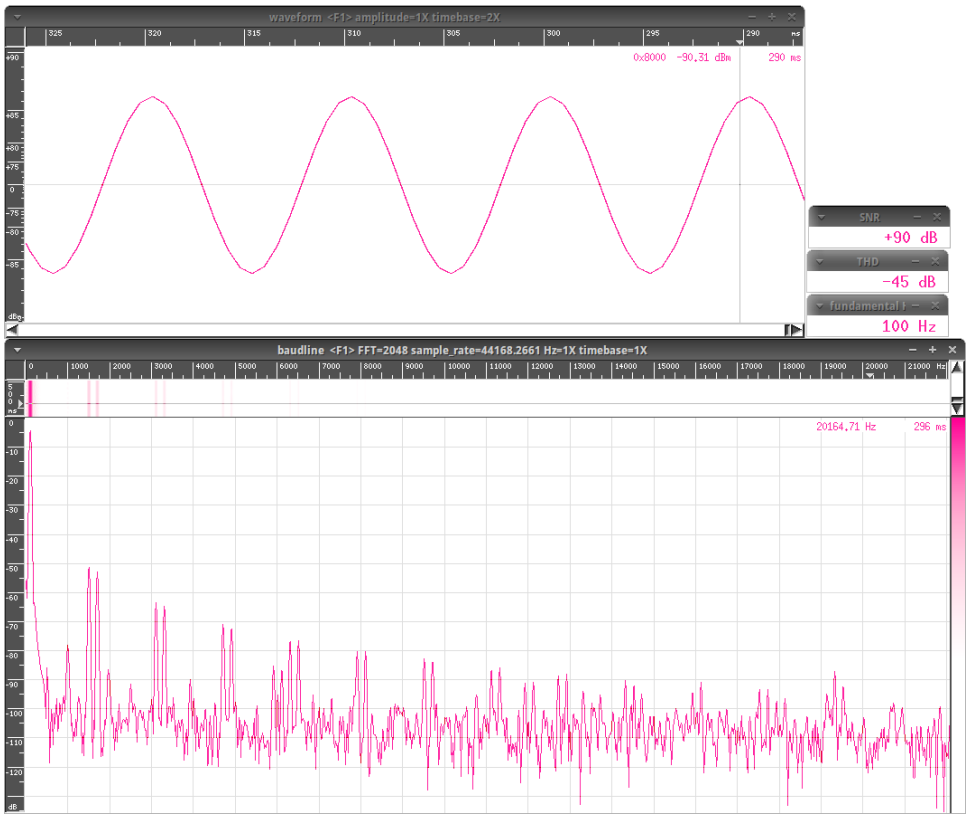


Fig. 52 – Signal généré avec : Nbre d'échant. du sinus = 16 / Nbre de sinus = 1 / Fréq. = 100Hz

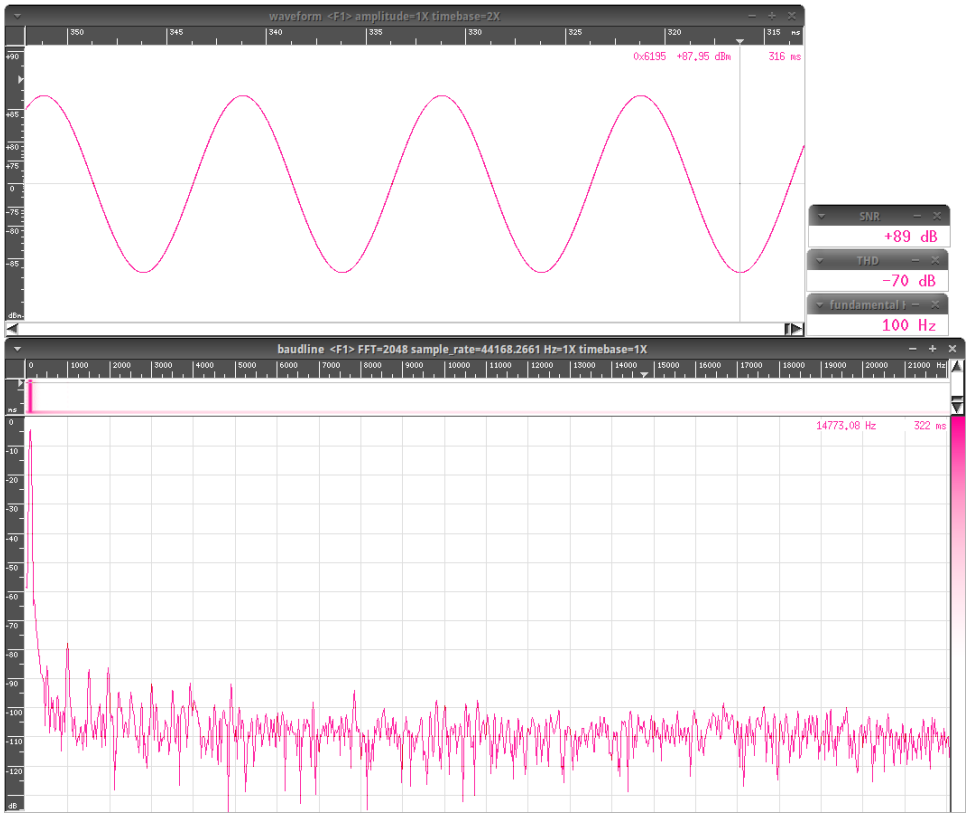


Fig. 53 – Signal généré avec : Nbre d'échant. du sinus = 1024 / Nbre de sinus = 1 / Fréq. = 100Hz

Le sinus de la *Fig. 52*, composé de 16 points, souffre encore de la production d'harmoniques non désirables à cause de l'interpolation linéaire, contrairement à celui de la *Fig. 53* dont l'effet est strictement négligeable, car généré avec un bien plus grand nombre de points (1024).

Fig. 54 – Signal généré avec : Nbre d'échant. du sinus = 4096 / Nbre de sinus = 4 / Fréq. = 440Hz

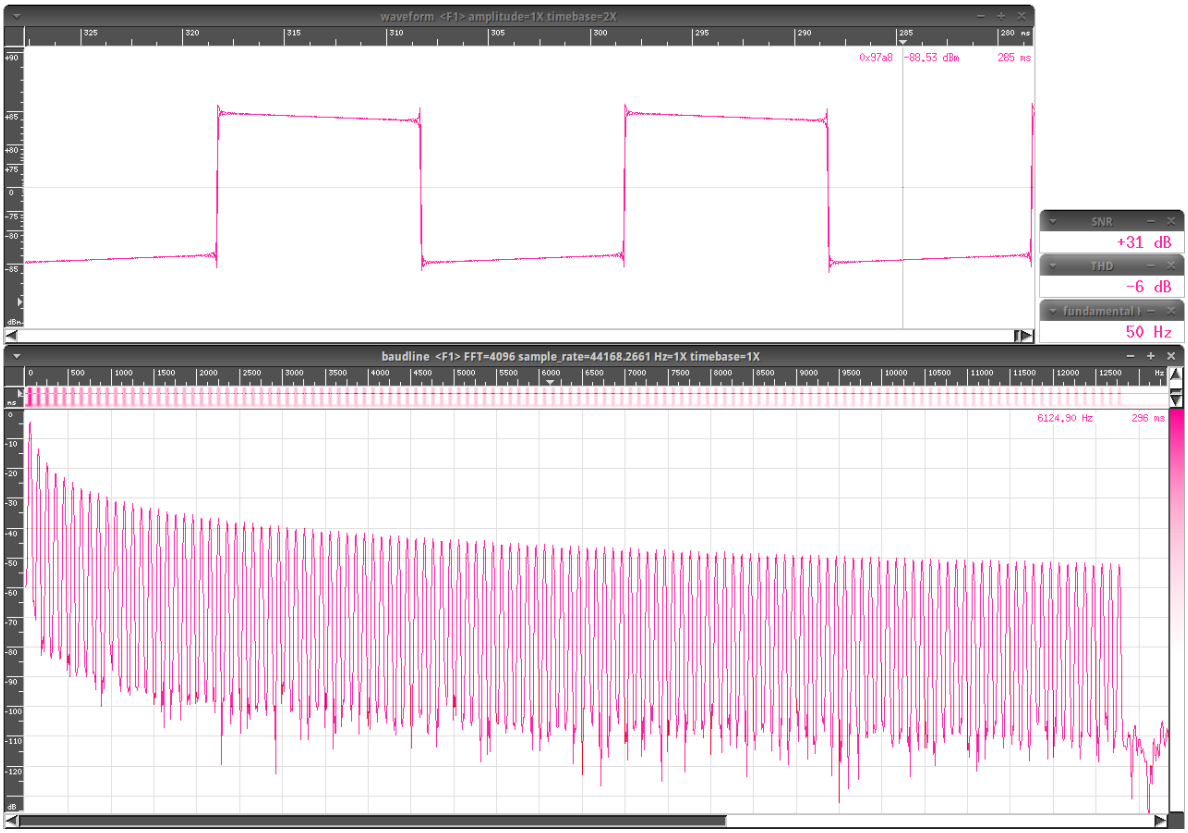


Fig. 55 – Signal généré avec : Nbre d'échant. du sinus = 1024 / Nbre de sinus = 128 / Fréq. = 50Hz

Les deux captures de cette page présente la génération d'un carré par la lecture simultanée de sinus (un sinus par canal logique).

Ces mesures prouvent donc le bon fonctionnement du SPU pour une utilisation basique.

4.4 TEST GLOBAL DU SYNTHETISEUR MIDI « SQUARE HARP »

Il s'agit du test final avec le synthétiseur « SquareHarp » de démonstration. Voici les différents essais effectués et leurs résultats :

ORDRE	DESCRIPTION DU TEST	SUCCÈS
#1	**Relier la board au PC par RS232 (connexion MIDI). Relier également la sortie audio sur l'entrée du PC en vue d'analyser le son. Configurer le code pour un rendu sur au maximum 192 canaux avec une taille de paquet de 64 échantillons. Programmer la FPGA et exécuter le code du CPU.** Le texte « HSPU » doit s'afficher sur les 7-segments.	✓
#2	**Ouvrir un fichier MIDI avec un quelconque séquenceur MIDI et le paramétrer pour qu'il joue sur la sortie RS232 (nécessite l'installation du driver YAMAHA CBX-MIDI). Lancer la lecture. Attendre un moment, écouter. Stresser le système en augmentant le tempo à des valeurs abusives (>1000bpm).** Les LEDs rouges indiquent le nombre d'événements MIDI reçus et joués. La board génère bien du son en sortie (son volume est réglable à l'aide des deux boutons bleus de droite) et l'on peut voir les LEDs vertes se comporter comme des vumètres. Le son en sortie est en stéréo et ne présente aucun défaut. La hauteur des notes ne diminue pas subitement après quelques secondes (signe que le SPU serait en surcharge). Le SPU est insensible à des *bursts* très important d'événements MIDI.	✓
#3	**Observer l'allure du son via l'entrée de la carte son du PC ou à l'oscilloscope.** Rien n'est anormal lors d'une analyse temporelle et spectrale.	✓

Tabl. 6 – Liste des tests effectués avec le synthétiseur « SquareHarp »

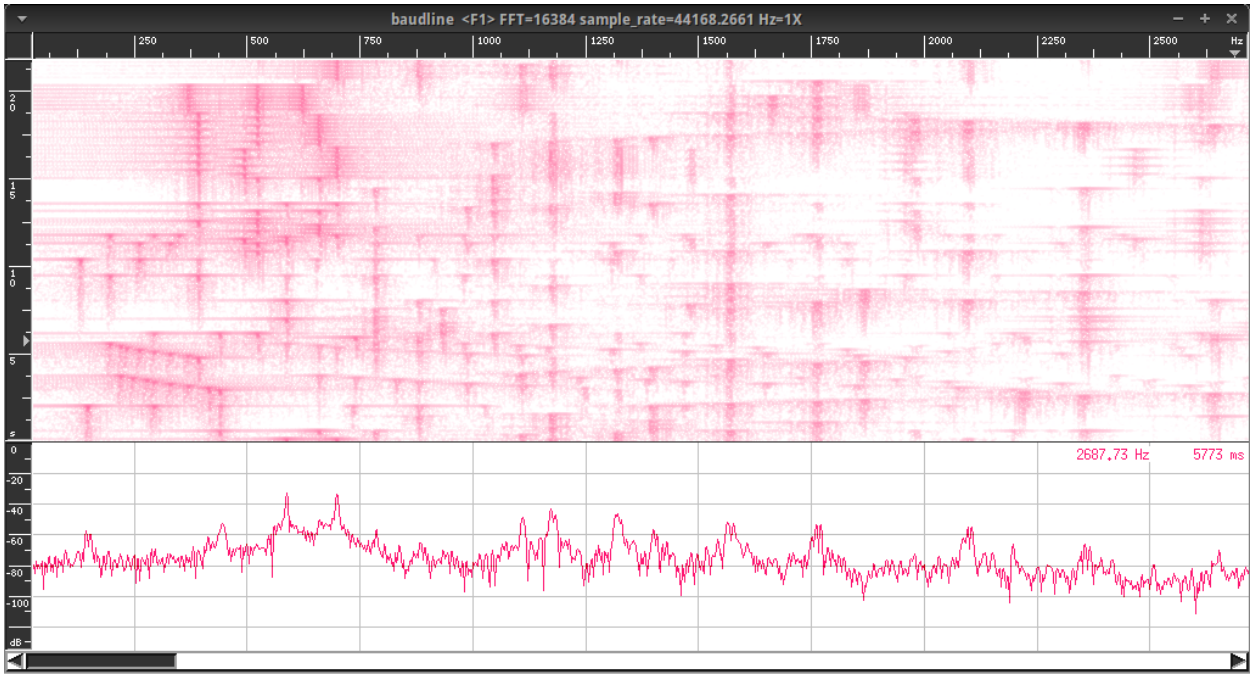


Fig. 56 – Analyse spectrale à la sortie du D/A lorsque le synthétiseur joue une mélodie

Des tests additionnels ont également été effectués jusqu'à 256 canaux logiques actifs et ont permis de déterminer par expérimentation une approximation de la taille des paquets rendus par le SPU en fonction du nombre de canaux logiques actifs simultanément (avec une marge d'environ x1.5) :

$$packetSize = \left(\frac{channelCount}{\max ChannelCount} \right) \cdot 128 = \left(\frac{channelCount}{256} \right) \cdot 128 = \left(\frac{channelCount}{2} \right)$$

La latence du SPU dans le pire des cas entre le moment où le CPU finit de (re)paramétrer un canal logique et son résultat sur le convertisseur D/A est exprimable ainsi :

$$latency_{MAX} = \frac{packetSize + dacFifoDepth}{f_{MIXING}} = \frac{packetSize + 256}{88'200}$$

Sachant que la taille d'un paquet est comprise entre 1 et 256 échantillons, cette latence sera entre **2.9ms** et **5.8ms**. La diminution de la taille de la FIFO du D/A est tout à fait envisageable pour obtenir des latences encore plus faibles.

Néanmoins, il est à noter que ce calcul ne correspond pas à la latence totale 'perçue' par l'utilisateur. Viendront également s'ajouter :

- La latence du périphérique contrôlant le système par MIDI.
- La latence du CPU pour la conversion des événements MIDI en ordre pour le SPU.
- La latence du système d'amplification audio.

5. AMELIORATIONS FUTURES

Dans l'état actuel, le SPU possède tous les ingrédients de base pour pouvoir être exploité à des fins musicales. Son architecture a l'avantage d'être très flexible, puisqu'elle supporte sans autre l'ajout de nouvelles unités au canal physique – sans remise en question intégrale de la structure. Néanmoins, certaines fonctionnalités lui font encore défaut…

5.1 DANS UN FUTUR PROCHE

Actuellement, il manque encore quelques éléments au SPU pour une accélération hardware de base complète. L'une des unités faisant le plus défaut est un générateur d'enveloppe contrôlant le volume du canal, en plus de la (semi-)constante de volume de l'utilisateur :

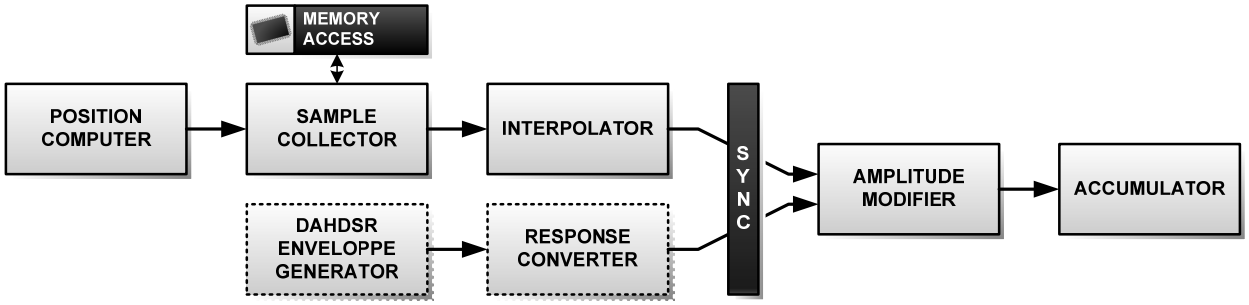


Fig. 57 – Canal physique de rendu sonore amélioré par une enveloppe contrôlant le volume du canal

5.1.1 UNITÉ « DAHDSR ENVELOPPE GENERATOR »

En général, on cherche à ce que les enregistrements sonores soient de courte durée pour des raisons d'économie d'espace mémoire. Pour cela, on se limite souvent à l'attaque et au premier motif périodique du maintien de l'instrument enregistré, voire seulement à son maintien.

Tel que le système est réalisé actuellement, le seul moyen de modifier l'amplitude sonore dans le temps est que le CPU modifie le registre du volume gauche/droit des canaux logiques concernés à intervalles réguliers. Or, cette méthode possède quelques inconvénients :

- La granularité de modification du volume se limite à la taille d'un paquet. Si la différence d'amplitude entre deux changements est particulièrement abrupte, la qualité du rendu sonore se verra dégradée.
- Le CPU sera rapidement amené à saturation dans le cas d'une actualisation très fréquente de nombreux canaux logiques. Le SPU sera également ralenti si le CPU ne verrouille et ne déverrouille pas suffisamment vite les canaux logiques.

La solution à ces problèmes est d'accélérer directement en hardware cette fonctionnalité en tant que nouvelle unité du canal physique de rendu.

Il existe plusieurs méthodes pour générer une modulation d'enveloppe en amplitude.

L'une d'entre elles consiste à la stocker sous forme d'échantillon dans la mémoire de travail du SPU et de la lire exactement de la même manière que sont lus les échantillons audio. Cependant, cette technique est coûteuse en ressources, car elle demande de dupliquer les unités « Position Computer », « Sample Collector » et « Interpolator ». Elle nécessite ainsi un accès à la mémoire de travail du SPU.

Une autre méthode, répandue[19], consiste en l'approximation du comportement de l'amplitude dans le temps en la réduisant à un motif spécifique paramétrable, composé au minimum d'une attaque et d'un relâchement. Un motif se prêtant bien à une utilisation générique est l'enveloppe de type "DAHDSR" (**D**elay, **A**ttack, **H**old, **D**ecay, **S**ustain, **R**elease). Elle est caractérisée par 6 phases exprimées par différents paramètres :

[19] Infos complémentaires : *http://en.wikipedia.org/wiki/Synthesizer#ADSR_envelope*

- **Phase de délai (*Delay*)** ⅄
  Caractérisée par un temps durant lequel l'amplitude est nulle (temps de délai / *delay time*).
- **Phase d'attaque (*Attack*)** ⅄
  Caractérisée par la transition de l'amplitude nulle à l'amplitude maximale en un certain temps (Temps d'attaque / *Attack time*).
- **Phase de maintien (*Hold*)** ⅄
  Caractérisée par le temps durant lequel l'amplitude reste maximale (Temps de maintien / *Hold time*).
- **Phase de déclin (*Decay*)** ⅄
  Caractérisée par la transition de l'amplitude maximale à l'amplitude de maintien en un certain temps (Temps de déclin / *Decay time*).
- **Phase maintien/soutien (*Sustain*)**
  Caractérisée par une amplitude constante à la valeur de l'amplitude de maintien (*sustain*) tant que la note est maintenue. (Amplitude de maintien, soutien / *Sustain amplitude*)
- **Phase de relâchement (*Release*)**
  Caractérisée par la transition de n'importe quelle amplitude à une amplitude nulle.

Les phases marquées par le symbole « ⅄ » peuvent être configurées individuellement pour directement passer à la phase de relâchement dans le cas où la note serait relâchée avant d'être dans la phase de maintien (*sustain*). Le passage de la phase de maintien (*sustain*) à celle de relâchement (*release*) est implicite.

L'illustration suivante nous montre la représentation graphique de l'enveloppe :

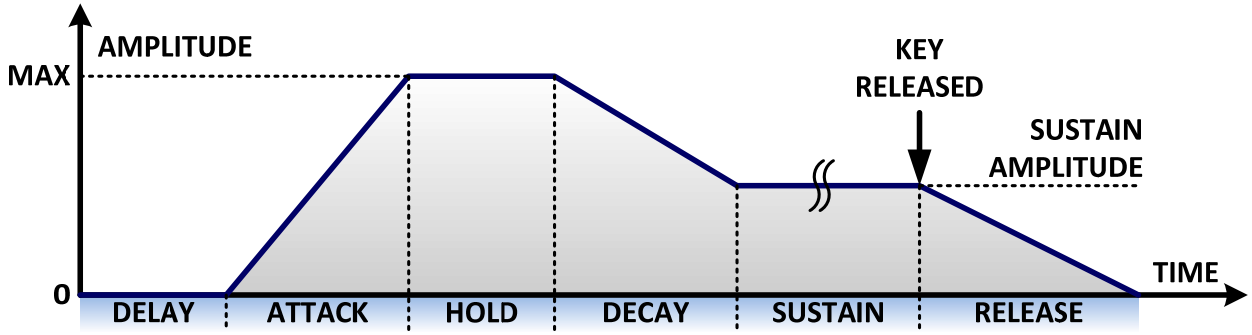


Fig. 58 – Enveloppe de type DAHDSR

Néanmoins, cette enveloppe ne sera pas reproduite exactement à chaque note jouée. En effet, si la note est relâchée avant d'être dans la zone de maintien (sustain), l'enveloppe générée pourra se comporter de diverses manières selon la configuration de l'acceptance de la transition des diverses phases vers la phase de relâchement (release). Ceci est illustré à la figure suivante :

ENVELOPPE BEHAVIOUR DESCRIPTION

A POSSIBLE CASE (TIME SIMULATION)

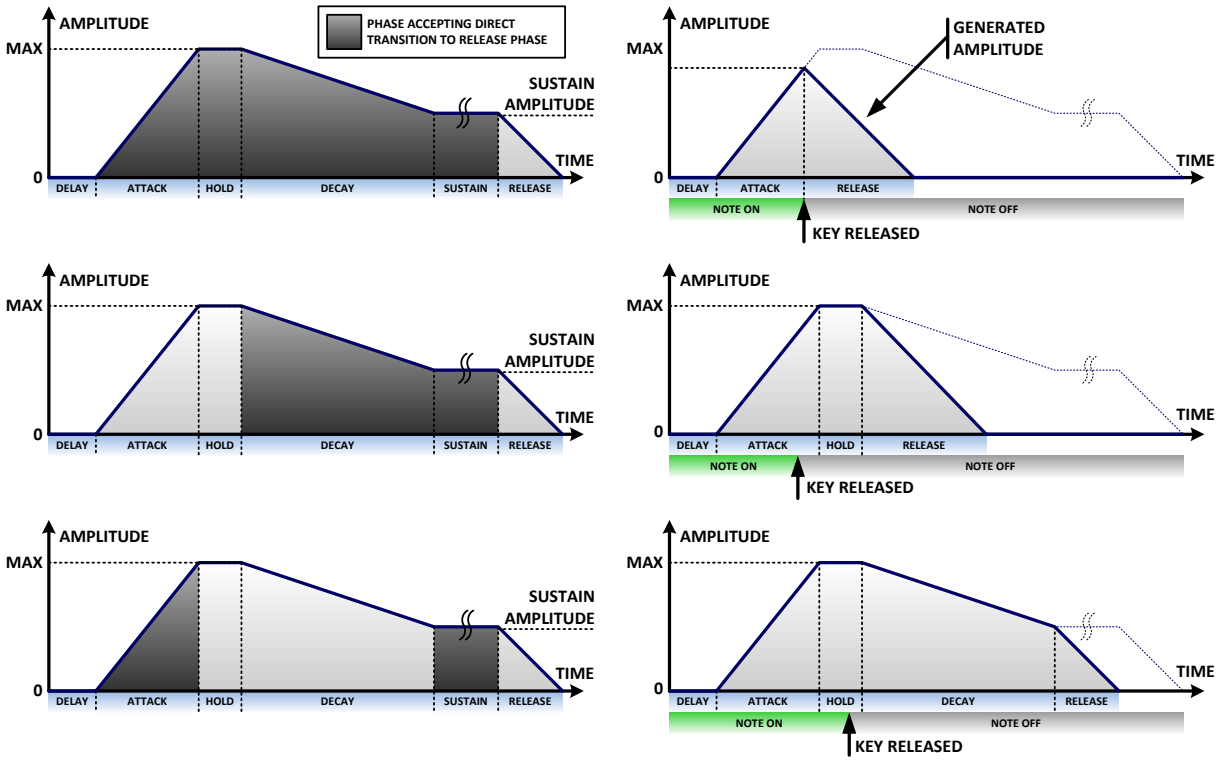
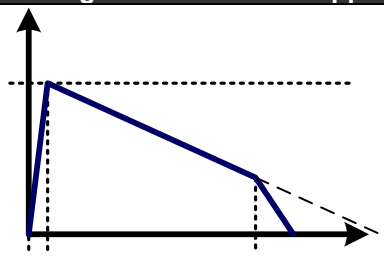
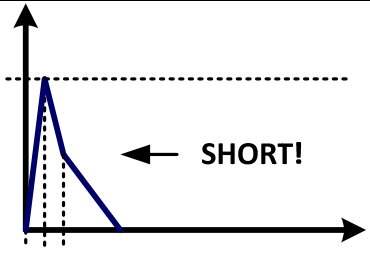
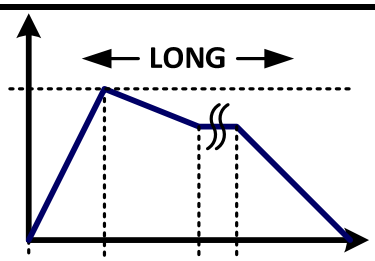


Fig. 59 – Gestion du relâchement de la note dans l'enveloppe DAHDSR

Voici quelques exemples d'utilisation d'une telle enveloppe routée sur le volume du canal :

Configuration de l'enveloppe	Exemple d'utilisation
 **Delay** : *None* **Attack** : Very f*ast* **Hold** : *None,* **Decay** : *Slow,* **Sustain** : *Set to zero* **Release** : F*ast*	Voici une enveloppe typique reproduisant le comportement en amplitude d'une touche de piano (acoustique). L'attaque est rapide et le temps de déclin particulièrement long. Il n'y a pas de temps de délai, ni de maintien à l'amplitude maximale. La valeur de maintien est nulle afin que, lorsque la touche est maintenue longtemps, l'amplitude puisse atteindre zéro. Aussitôt la touche relâchée, on passe immédiatement dans la phase de relâchement, dont la pente est bien plus raide.
 **Delay** : *None* **Attack** : Very f*ast !* **Hold** : *None* **Decay** : Very f*ast !* **Sustain** : *1/2 max. amplitude* **Release** : F*ast*	Cette enveloppe trouverait usage pour des instruments percussifs (p.ex. : Eléments d'une batterie comme charleston, caisse claire). Toutes les pentes, à savoir celles de d'attaque, de déclin et de relâchement, sont très rapides. La pente de relâchement est néanmoins moins importante que celle de déclin. Le placement de la valeur de maintien à la moitié de l'amplitude maximale de l'enveloppe permet de laisser sonner un peu plus la fin du son produit. On suppose également que ces instruments ne seront pas réceptifs au maintien des touches (dans le cas d'un clavier). Dans le cas contraire un maintien non désirable à la moitié de l'amplitude apparaîtrait. Cela nécessite néanmoins que la phase de déclin soit maintenue malgré l'absence de maintien.

 **Delay** : *None,* ⅄ **Attack** : Slow, ⅄ **Hold** : *None,* ⅄ **Decay** : Very slow, ⅄ **Sustain** : *¾ max. amplitude* **Release** : Slow	L'enveloppe que voici possède des temps de montées/descentes particulièrement long. Elle trouverait usage pour réaliser un « synth pad » (instrument à attaque et relâchement long, utilisé en soutien/arrière plan d'une musique).

⅄ *Ces phases sont réceptives au relâchement de la note. Dans un tel cas on finirait immédiatement dans la phase de relâchement (Release).*

Tabl. 7 – Exemple d'utilisation d'une enveloppe contrôlant le volume d'un canal logique

Le schéma suivant exprime visuellement l'incidence réalisée sur le son par cette unité, contrôlant le volume du canal :

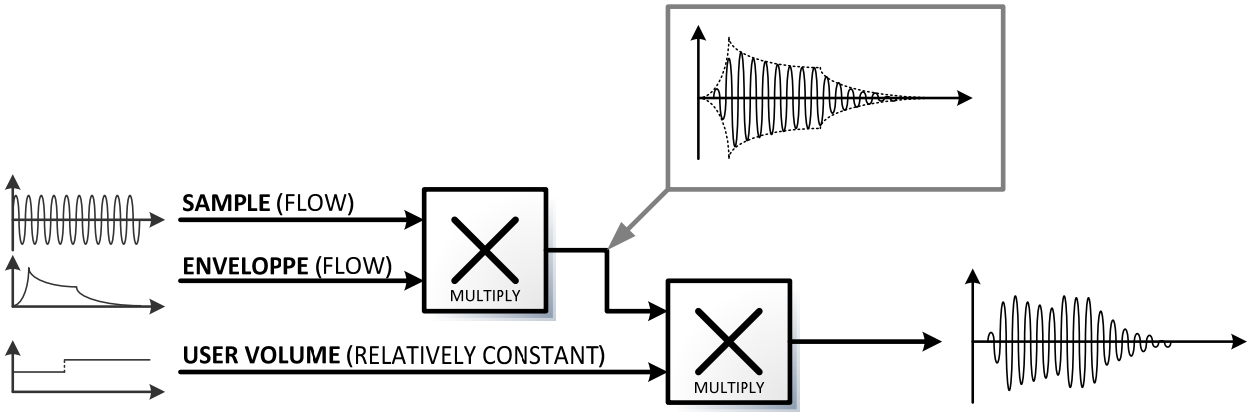
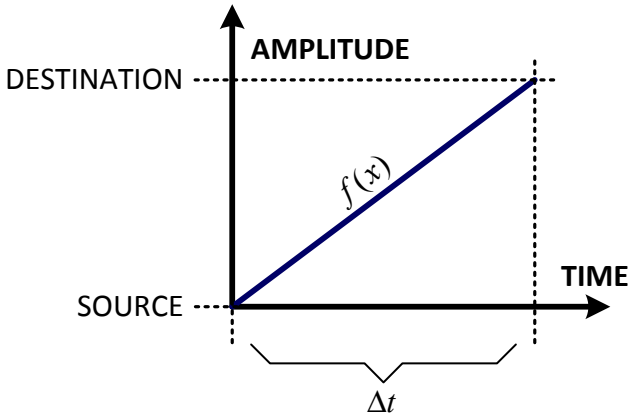


Fig. 60 – Résultat de l'utilisation d'une modulation d'enveloppe mappée sur le volume d'un canal[20]

5.1.1.1.1 Réalisation numérique

Pour une implémentation dans la FPGA, il est plus simple de travailler avec des vitesses de montée/descente de l'enveloppe selon la phase dans laquelle on se situe (Decay, Sustain, etc…), bien qu'il soit plus logique pour l'utilisateur de réfléchir en temps (plus coûteux en calculs, donc en ressources). La vitesse de l'évolution de l'amplitude dans une phase n'est rien d'autre que sa pente :

[20] L'enveloppe présentée a un comportement exponentiel. Voir unité « Response Converter ».



$$speed = f'(x) = \frac{A_{DESTINATION} - A_{SOURCE}}{\Delta t}$$

Fig. 61 – Pente de l'amplitude d'une phase de l'enveloppe

Le schéma en annexe « SCH1 – Page 1 » présente une structure d'implémentation de l'unité basée sur ce concept.

Le driver du SPU pourrait gérer nativement cette expression de l'enveloppe, mais aussi offrir la possibilité d'interpréter des enveloppes exprimées en temps et non en vitesse.

5.1.2 UNITÉ « RESPONSE CONVERTER »

Le contrôle linéaire du volume d'un canal (c'est-à-dire avec une valeur comprise entre 0 et 2^n) n'aura pas un comportement linéaire aux oreilles de l'auditeur (dont la perception est logarithmique). En effet, si l'on contrôle sous forme de rampe le volume d'un canal logique jouant un sinus, l'utilisateur le percevra comme subissant une brutale montée du volume suivie d'une augmentation progressive de plus en plus légère.

Pour obtenir un contrôle du volume – en apparence – linéaire qui produit une réponse également linéaire à l'oreille humaine, le driver du SPU qu'utilise le CPU aurait pu simplement corriger les amplitudes via une table de conversion. Mais cela n'aurait pu s'appliquer qu'au contrôle direct du volume absolu des canaux logiques et non au générateur d'enveloppe contrôlant également le volume du canal. La figure suivante illustre bien le cas :

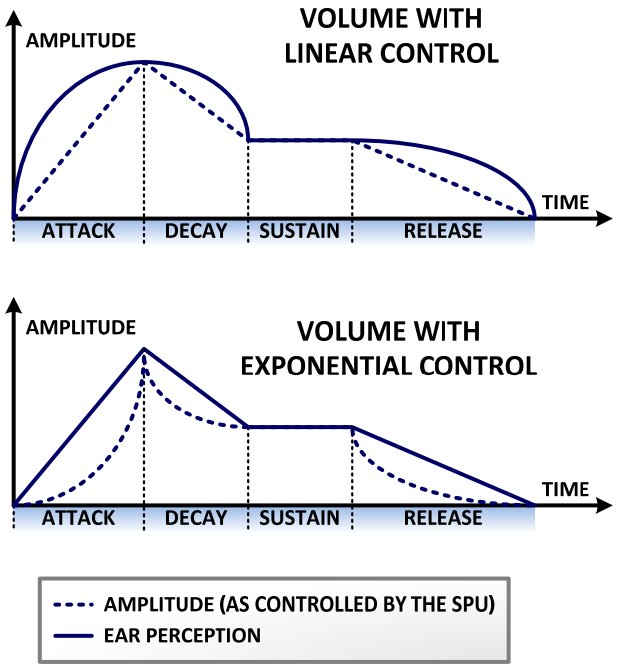


Fig. 62 – Perception de l'oreille humaine selon un contrôle linéaire ou exponentiel du volume

Pour cela, il est nécessaire d'ajouter une unité effectuant la conversion de notre expression linéaire du volume. L'équation suivante présente la pression sonore en dB à partir d'une pression *p* :

$$L_P[dB] = 10 \cdot \log_{10}\left(\frac{p^2}{p_{REF}{}^2}\right) = 20 \cdot \log_{10}\left(\frac{p}{p_{REF}}\right)$$

Aussi, dans sa norme DLS1, la 'MIDI Manufacturers Association' définit le comportement de l'amplitude sonore selon [21] :

$$gain[dB] = 20 \cdot \log_{10}\left(\frac{Volume^2}{127^2}\right) \qquad Avec: 0 \le Volume < 128$$

A partir de ces deux équations, on peut déduire par analogie une équation permettant de générer une table de conversion prenant en entrée le volume exprimé entre 0 et $2^{15}-1$ et fournissant en sortie le comportement incurvé pour une échelle également comprise entre 0 et $2^{15}-1$:

$$output = \left(\frac{input^2}{\left(2^{bit\ count}\right)^2}\right) \cdot \left(2^{bit\ count}\right)$$

La courbure de cette fonction peut être augmentée en modifiant la valeur des puissances :

$$output = \left(\frac{input^n}{\left(2^{bit\ count}\right)^n}\right) \cdot \left(2^{bit\ count}\right) \ , \ 1 < n$$

L'unité « Amplitude Modifier » consommerait des volumes exprimés en 16bits signés. Pour offrir à l'utilisateur une plus grande liberté, cette unité embarquerait 4 demi-tables de conversion (partie positive uniquement, 512 points, avec interpolation linéaire). Voici le plot de chacune de ces demi-tables :

[21] Source : *http://www.midi.org/techspecs/dls/dls1v11b.pdf – Page 14 et 57*

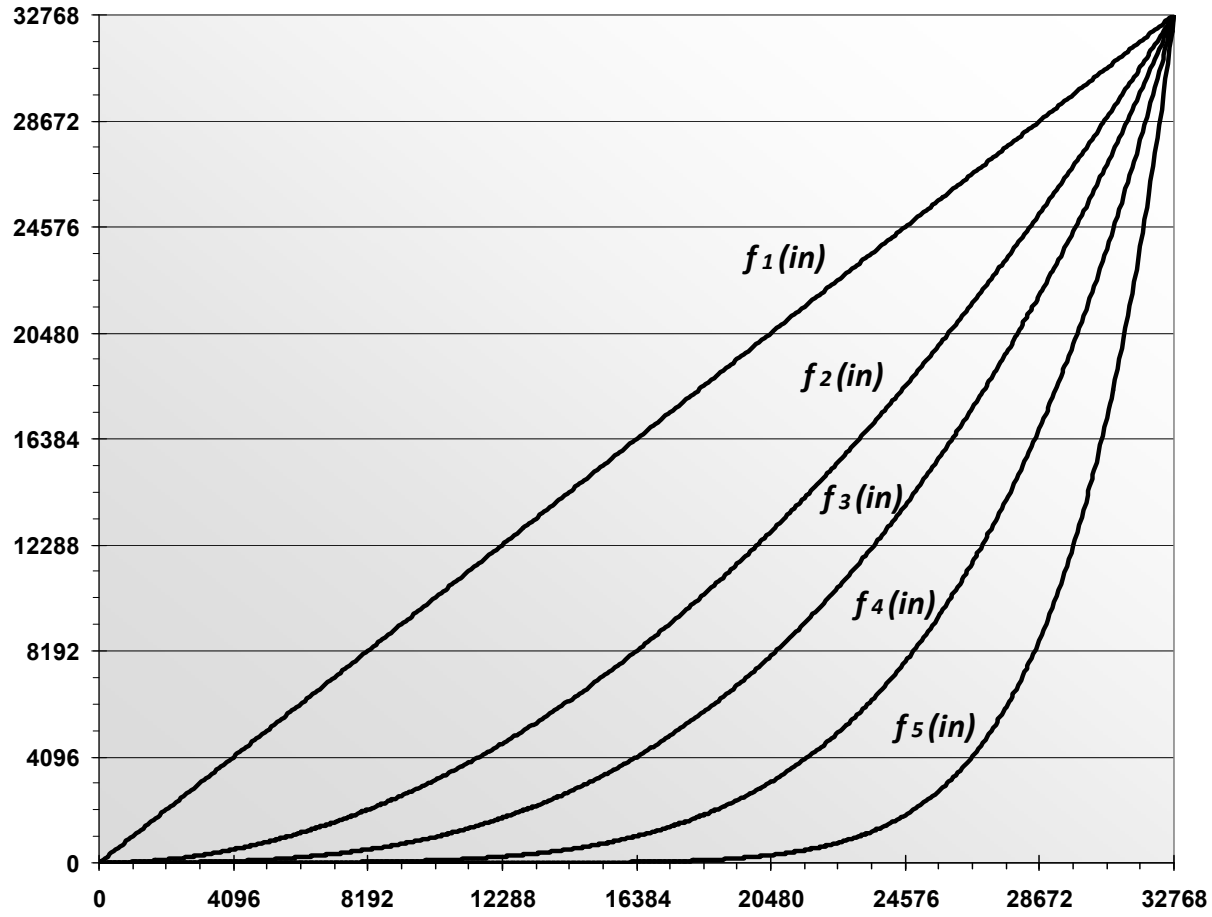


Fig. 63 – Plot des tables de conversion de l'unité « Response Converter »

Avec :

Fonction	Remarque sur la fonction
$f_1(in) = in$	L'entrée est directement transmise à la sortie (bypass).
$f_2(in) = \left(\frac{in^2}{\left(2^{15}\right)^2} \right) \cdot \left(2^{15}\right)$	Cette fonction produit une réponse qui correspond aux spécifications de la 'MIDI Manufacturers Assossiation' (norme DLS1).
$f_3(in) = \left(\frac{in^3}{\left(2^{15}\right)^3} \right) \cdot \left(2^{15}\right)$	Réponse un peu plus marquée que la précédente (f_2).
$f_4(in) = \left(\frac{in^5}{\left(2^{15}\right)^5} \right) \cdot \left(2^{15}\right)$	-
$f_5(in) = \left(\frac{in^{10}}{\left(2^{15}\right)^{10}} \right) \cdot \left(2^{15}\right)$	Cette fonction produit une réponse d'apparence très linéaire à l'oreille humaine, mais souffre d'une inutilisation d'un tiers de la plage des valeurs d'entrée (0 à ~11'000) dont la valeur de sortie est nulle (cela pourrait être résolu par une augmentation de la résolution du volume de 16bits à 24, voire 32bits). A utiliser dans des cas spécifiques.

Tabl. 8 – Fonctions des demi-tables de l'unité « Response Converter »

Le schéma « *SCH1* – Page 2» en annexe présente une structure d'implémentation possible de cette unité.

5.1.3 SYNCHRONISATEUR D'UNITÉS

Les deux unités précédemment présentées produisent un flux de donnée qui a besoin d'être synchronisé avec l'arrivée des échantillons sur l'unité « Amplitude Modifier » (selon *Fig. 57*). Ainsi, la présence d'un arbitre pour la synchronisation des données est nécessaire. Le schéma en annexe « *SCH1* – Page 3» présente une solution d'implémentation simple pour ce problème.

5.2 DANS UN FUTUR UN PEU PLUS LOINTAIN

Le rajout de certaines unités additionnelles ferait de ce SPU un accélérateur hardware particulièrement complet. Le schéma bloc ci-dessous présente ces unités dans le canal physique.

5.2.1 UNE ACCÉLÉRATION PLUS COMPLÈTE DU CONTRÔLE DU VOLUME

Malgré la présence de l'enveloppe DAHDSR contrôlant le volume du canal, il manque encore la possibilité de pouvoir effectuer des tremolos de manière hardware. L'ajout d'une unité « LFO » en parallèle du générateur d'enveloppe solutionnerait ce problème. Le résultat de ces deux unités serait ensuite additionné (protection contre les débordements nécessaire) avant d'être multiplié sur l'échantillon pour en altérer l'amplitude.
Il est à noter que le « LFO » doit offrir la possibilité de régler la fréquence, l'amplitude et la forme d'onde.

5.2.2 PROBLÉMATIQUE DES VIBRATOS ET DES PORTAMENTOS

Jusque là, seules les altérations d'amplitude ont été considérées. Pourtant, les altérations de hauteur nécessitent la mise en place de mécanismes similaires :

La génération d'un vibrato est effectuable avec l'unité « LFO ».

L'application systématique de portamentos sur l'attaque et/ou le relâchement pourrait être réalisée avec une unité « 3 Phase Ramp Generator » générant une enveloppe signée composée de 3 phases :

- **Phase #1**
  Déplacement de la valeur initiale vers 0 à une vitesse spécifiée.
- **Phase #2**
  Attente du relâchement de la note.
- **Phase #3**
  Déplacement de la valeur 0 vers une valeur de destination à une vitesse spécifiée.

Le résultat de ces deux unités et la constante utilisateur de hauteur passerait à travers un additionneur pour obtenir la représentation linéaire instantanée de la hauteur du son. Celle-ci serait ensuite transformée à travers une nouvelle unité « Linear Expression to Speed Convertor » vers une vitesse de lecture.

Il est à noter que ces valeurs linéaires instantanées seraient exprimées avec une finesse de 1/128ème de ton et non pas par cent (1/100^{ièmes} de demi-tons) dans le but de simplifier sensiblement l'implémentation du « Linear Expression to Speed Convertor » qui pourrait reposer sur l'utilisation de deux ROMs (demi-tons et 1/128^{ième} de demi-tons).

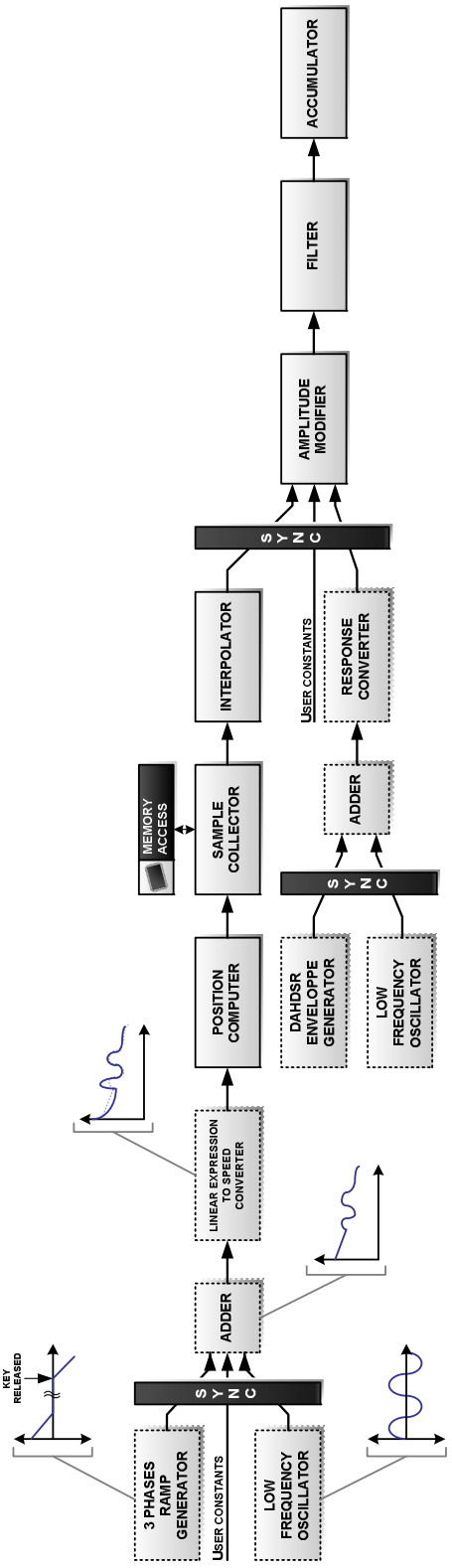


Fig. 64 – Proposition de version étendue du canal physique

5.2.3 FILTRES

Le son que l'on peut produire est actuellement trop « parfait ». Il manque la possibilité de pouvoir l'altérer un minimum. Un filtre passe-bas résonnant de 2ème ordre serait idéal dans ce sens : en effet, il est couramment utilisé avec des sons issus de synthèse pour en altérer la clarté. Tant la fréquence que la résonance doivent être contrôlables, de préférence par une enveloppe. L'ajout d'un filtre passe-haut pourrait également être la bienvenue.

5.2.4 FORMATS DE « COMPRESSION » SIMPLES

L'usage du DPCM, voire du ADPCM serait une solution intéressante pour économiser de l'espace sur la mémoire de travail du SPU. Ils permettraient typiquement une réduction de 2 fois, respectivement, 4 fois la taille des échantillons en LPCM.

Toutefois, le ADPCM ne pourrait pas supporter de boucles bidirectionnelles et serait très probablement limité à un codage sur nibbles (4bits).

Dans tous les cas, ces formats entraineraient une complexification de l'unité « Sample Collector » qui devrait conserver une valeur absolue des échantillons situés sur les débuts des boucles, en plus du dernier échantillon décodé.

5.2.5 INTERPOLATION

L'interpolation linéaire convient bien en général, mais n'offre pas encore une qualité suffisante dans certains cas spécifiques, comme pour les fichiers lus à basse vitesse relativement à leur fréquence d'échantillonnage. Le rajout d'une interpolation par *splines* serait à considérer (nécessite 4 échantillons en entrée pour la production de l'échantillon de sortie).

5.2.6 NOMBRE DE SORTIES AUDIO GÉNÉRIQUE

Actuellement, le SPU ne produit un rendu que sur deux canaux (stéréo). Ceci est particulièrement limitant dans le cas où on désirerait appliquer des effets (réverbération, compresseur, etc…) via des racks externes dédiés ou un module hardware sur seulement quelques voix.

Une très bonne solution serait que le SPU génère un *stream* de sortie avec, par exemple, 8 ou 16 canaux dont l'amplitude serait contrôlable individuellement.

6. SYNTHESE

Actuellement, le projet HSPU, c'est la combinaison :

- …d'un SPU hardware implémenté dans une FPGA avec les caractéristiques suivantes :
  - Cadencé à 50MHz
  - Rendu à 88.2kHz 32bits stéréo (mais accepte n'importe quelle fréquence de mixage)
  - Polyphonie jusqu'à 256 voix simultanées
  - Faible latence (maximum ~6ms)
  - Mémoire RAM pour les échantillons audio de 2MB
  - Support d'échantillons audio LPCM 8/16/32bits, little/big endian, unsigned/signed
  - Prise en charge de deux boucles pour la lecture des échantillons : boucle de maintien (*sustain*, contrôlée par le maintien de la voix) et boucle de relâchement (*release/default*). Peuvent être configurées individuellement pour une lecture monodirectionnelle ou bidirectionnelle.
  - Interpolation de type Sample&Hold (au plus proche) et linéaire
  - Contrôle du volume individuel gauche et droit. Résolution de 16bits (signé, inversion de phase possible).
- …d'un CPU (softcore) équipé d'une interface MIDI (UART) dont la mission est de :
  - Collecter / contrôler l'intégrité / filtrer les événements MIDI reçus
  - Interpréter les événements MIDI pour les convertir en actions sur le SPU
  - Piloter le SPU

Voici la liste des objectifs du projet et leur état :

- **Etude d'une architecture hardware pour un synthétiseur pipeliné en co-design**
  Objectif atteint. L'architecture étudiée a l'avantage d'être performante et évolutive – l'ajout - ou la modification - de fonctionnalités ne remettrait pas en cause tout le design.
- **Réalisation et tests du synthétiseur en VHDL**
  Objectif presque atteint. Certaines unités du canal physique de rendu sonore, soit le générateur d'enveloppe DAHDSR et le convertisseur de réponse pour le contrôle de l'amplitude sonore, ont été étudiées, mais n'ont pu être réalisées dans le temps imparti. Le développement d'une unité LFO a été mis de côté. Néanmoins, toutes les fonctionnalités de base pour le rendu sonore (à savoir : prédiction de la position en fonction de la vitesse, collection puis conversion des échantillons, interpolation, modification de l'amplitude et accumulation) ont été implémentées et testées avec succès.
- **Interfaçage du synthétiseur via MIDI**
  Objectif atteint. Le driver pour le CPU en charge de la réception, du contrôle et du décodage des trames MIDI est entièrement fonctionnel.
- **Réalisation d'un démonstrateur permettant de jouer du son à partir d'une interface MIDI**
  Objectif atteint. Le démonstrateur réalisé est un instrument MIDI de très bonne qualité (rendu stéréo avec réverbération) conçu pour une exploitation maximale des ressources du SPU. Il possède une polyphonie de 96 voix pour un rendu 32bits à 88.2kHz.

7. CONCLUSION

Le HSPU est un projet essentiellement expérimental réunissant les domaines de l'embarqué et de l'audio. Il a consisté en la réalisation d'un accélérateur hardware (dans une FPGA en VHDL) spécialisé dans la lecture massivement parallèle de fichiers audio échantillonnés, évitant ainsi l'utilisation d'un processeur cadencé à plusieurs centaines de MHz, remplacé par un petit processeur de pilotage (code en C++).

Son développement a été orienté dans le domaine musical en l'utilisant en tant que synthétiseur piloté par une interface MIDI. Cela a débouché sur la réalisation d'une structure flexible où une multitude de canaux audio logiques (virtuels) se font charger tour à tour dans un canal de rendu physique, composé d'unités de traitement chaînées en vue de produire le son.

Le temps disponible était malheureusement très limité pour pouvoir amener ce projet jusqu'à un état avancé. Ainsi, les développements relatifs aux accès mémoire ont été plus gourmands en temps que prévu. C'est pourquoi, le développement de certaines unités (non vitales, mais importantes) n'a pu être achevé. Malgré tout, les objectifs fixés ont été majoritairement atteints.

Les résultats obtenus se sont avérés à la hauteur des attentes et sont très prometteurs quant à une future continuation du développement. En effet, la réalisation dans un premier temps d'un petit synthétiseur software à dent de scie a montré l'intérêt d'une accélération hardware en termes de performances. Aussi, la conception d'un simple générateur de carrés uniquement basé sur la lecture d'un sinus laisse entrevoir une utilisation du SPU en synthèse additive. Finalement, l'instrument MIDI de démonstration développé offre un bel exemple d'utilisation du système en tant qu'échantillonneur (*sampler*).

La polyphonie élevée (jusqu'à 256 voix), la faible latence (plus petite que 6ms) et la qualité du rendu (88.2kHz 32bits stéréo) font de ce SPU un accélérateur tout à fait adapté pour un usage musical.

8. DATE ET SIGNATURE

DATE : *Sion, le 9 juillet 2012*

SIGNATURE : *Alexandre Ganchinho*

9. GLOSSAIRE

ADPCM Abréviation de « Adaptative Differential Pulse Code Modulation ». Amélioration du DPCM par le rajout d'une notion d'échelle dynamique sur les différences d'amplitude. Permet typiquement une réduction de la largeur des échantillons par 4 (p.ex : 16bits LPCM → 4bits ADPCM).

ADSR Abréviation de « Attack Decay Sustain Release » désignant une d'enveloppe décrite par un temps d'attaque, de déclin, de maintien et de relâchement.

CPU Abréviation de « Central Processing Unit ».

DPCM Abréviation de « Differential Pulse Code Modulation ». Code la différence d'amplitude entre deux échantillons (contrairement au LPCM qui code la valeur absolue de l'amplitude). Permet typiquement une réduction de la largeur des échantillons de moitié (p.ex : 16bits LPCM → 8bits DPCM).

Echantillon Dans le domaine de l'audio : Valeur instantanée (point) d'une onde sonore. Désigne également « fichier audio » - pour limiter les confusions, ce terme n'est pas utilisé dans ce rapport sous cette signification.

Echantillonnage Prélèvement d'échantillons à intervalles réguliers.

FPGA Abréviation de « Field Programmable Gate Array », soit un circuit logique programmable.

Glissando Glissement continu ou discontinu d'une note à une autre.

LFO Abréviation de « Low frequency oscillator ».

LPCM Abréviation de « Linear Pulse Code Modulation ». Codec audio exprimant une onde sonore sous forme d'amplitudes (quantification linéaire) stockées à intervalles temporels réguliers.

Monophonie (Par abus de language:) Incapacité à jouer plusieurs notes simultanément. Le terme exact est "Monodie".

Panning Traduit par « Panoramique polyphonique ». Exprime la position spatiale d'un son.

Pitch Traduit par « hauteur ». Détermine la hauteur d'un son à l'écoute.

Polyphonie Capacité à jouer plusieurs voix simultanément.

Portamento Glissement continu d'une note à une autre.

Sample Traduit par « échantillon » (voir échantillon).

Sampling Traduit par « échantillonnage » (voir échantillonnage).

SPU Abréviation de Sound Processing Unit. Désigne une unité de traitement sonore hardware. Typiquement présent sur les cartes son, les claviers/racks synthétiseurs, etc…

Tremolo Variation périodique de l'intensité sonore d'une note autour d'une certaine valeur.

Verilog HDL Un langage de description hardware.

Vibrato Variation périodique de la hauteur d'une note autour d'une certaine valeur.

VHDL Abréviation de « VHSIC (Very High Speed Integrated Circuit) Hardware Description Language », soit un langage de description hardware.

Voix Synonyme de note.

10. TABLE DES ILLUSTRATIONS

Fig. 1 – Concept général du SPU 4
Fig. 2 – Synthétiseur analogique basique 5
Fig. 3 – Connectique DIN5 utilisée pour le protocole MIDI 6
Fig. 4 – Représentation d'un échantillon audio non compressé (LPCM, 16bits signé) 7
Fig. 5 – Allure d'un module (sous le logiciel OpenMPT) 9
Fig. 6 – Kit de développement FPGA «Terasic DE2-70» 10
Fig. 7 – Extrait des caractéristiques des différents softcores NiosII 12
Fig. 8 – Schéma bloc d'utilisation du SPU 13
Fig. 9 – Architecture générale 13
Fig. 10 – Canal audio logique 14
Fig. 11 – Structure générale du système 15
Fig. 12 – Concept d'implémentation du contrôleur SSRAM 16
Fig. 13 – Structure d'un port du multiplexeur d'accès à la mémoire de travail 17
Fig. 14 – Canal physique de rendu sonore réalisé par le chaînage des unités de traitement sonore 18
Fig. 15 – Concept : Vitesse de lecture des échantillons en mémoire RAM 18
Fig. 16 – Concept : Collection et conversion des échantillons 19
Fig. 17 – Concept : Interpolation 19
Fig. 18 – Concept : Modification de l'amplitude des échantillons 19
Fig. 19 – Concept : Accumulateur d'échantillons pour la formation du paquet 19
Fig. 20 – Principe de chaînage des unités 20
Fig. 21 – Diagramme de timing de la communication entre les unités 20
Fig. 22 – Positions indexées dans un fichier audio 21
Fig. 23 – Boucle monodirectionnelle & bidirectionnelle 22
Fig. 24 – Diagramme de timing du bus Avalon-MM 25
Fig. 25 – Formatage de l'accès aux registres de configuration/contrôle du SPU par le CPU 26
Fig. 26 – Formatage de l'accès aux registres des canaux logiques par le CPU 27
Fig. 27 – Formatage de l'accès à la mémoire de travail du SPU par le CPU 28
Fig. 28 – Représentation graphique du rendu d'un paquet 28
Fig. 29 – Illustration du problème de l'interaction simultanée du CPU et du SPU sur un canal logique 29
Fig. 30 – Procédure de modification d'un canal actif par le CPU 29
Fig. 31 – Interface de contrôle du codec audio 30
Fig. 32 – Structure générale du CPU 31
Fig. 33 – Connexion CPU⇆SDRAM via un bridge (adaptation du domaine horloge) 31
Fig. 34 – Connexion CPU⇆SPU via un bus Avalon 32
Fig. 35 – Connexion CPU⇆Périphériques via un bridge 32
Fig. 36 – Mapping mémoire 33
Fig. 37 – Synthétiseur sawtooth : Comportement 34
Fig. 38 – Synthétiseur sawtooth : Comportement du « SawtoothChannel » 34
Fig. 39 – Diagramme de classe UML du synthétiseur sawtooth 35
Fig. 40 – « Square Generator » : Comportement 36
Fig. 41 – Diagramme de classe UML du « Square generator » 36
Fig. 42 – Instrument « SquareHarp » sous le logiciel de synthèse sonore ZynAddSubFX 37
Fig. 43 – Mappage des notes échantillonnées de l'instrument « SquareHarp » 38
Fig. 44 – Diagramme de flux du code de l'instrument « SquareHarp » 38
Fig. 45 – Diagramme de classe UML du synthétiseur MIDI «SquareHarp» 40
Fig. 46 – Extrait de simulation sous ModelSim de l'unité « Position Computer » 41
Fig. 47 – Extrait de simulation sous ModelSim de l'unité « Interpolator » 41
Fig. 48 – Extrait de simulation sous ModelSim de l'unité « Amplitude Modifier » 42
Fig. 49 – Analyse de l'activité sur la SSRAM 42
Fig. 50 – Jeu d'une mélodie polyphonique sur le synthétiseur soft Sawtooth 43
Fig. 51 – Signal généré avec : Nbre d'échant. du sinus = 4 / Nbre de sinus = 1 / Fréq. = 100Hz 44
Fig. 52 – Signal généré avec : Nbre d'échant. du sinus = 16 / Nbre de sinus = 1 / Fréq. = 100Hz 45
Fig. 53 – Signal généré avec : Nbre d'échant. du sinus = 1024 / Nbre de sinus = 1 / Fréq. = 100Hz 45
Fig. 54 – Signal généré avec : Nbre d'échant. du sinus = 4096 / Nbre de sinus = 4 / Fréq. = 440Hz 46
Fig. 55 – Signal généré avec : Nbre d'échant. du sinus = 1024 / Nbre de sinus = 128 / Fréq. = 50Hz 46
Fig. 56 – Analyse spectrale à la sortie du D/A lorsque le synthétiseur joue une mélodie 47
Fig. 57 – Canal physique de rendu sonore amélioré par une enveloppe contrôlant le volume du canal 49

Fig. 58 – Enveloppe de type DAHDSR 50
Fig. 59 – Gestion du relâchement de la note dans l'enveloppe DAHDSR 51
Fig. 60 – Résultat de l'utilisation d'une modulation d'enveloppe mappée sur le volume d'un canal 52
Fig. 61 – Pente de l'amplitude d'une phase de l'enveloppe 53
Fig. 62 – Perception de l'oreille humaine selon un contrôle linéaire ou exponentiel du volume 53
Fig. 63 – Plot des tables de conversion de l'unité « Response Converter » 55
Fig. 64 – Proposition de version étendue du canal physique 57

Tabl. 1 – Signaux d'un port du multiplexeur d'accès à la mémoire de travail 17
Tabl. 2 – Présentation des boucles de maintien (sustain) et de relâchement(release) / de défaut 22
Tabl. 3 – Liste des registres configuration/contrôle du SPU 26
Tabl. 4 – Liste des registres des canaux logiques 28
Tabl. 5 – Liste des tests effectués avec le synthétiseur soft Sawtooth 43
Tabl. 6 – Liste des tests effectués avec le synthétiseur « SquareHarp » 47
Tabl. 7 – Exemple d'utilisation d'une enveloppe contrôlant le volume d'un canal logique 52
Tabl. 8 – Fonctions des demi-tables de l'unité « Response Converter » 55

11. BIBLIOGRAPHIE

[1] Altera Corporation, *Documentation*
http://www.altera.com/literature/lit-index.html

[2] Dave Benson. *Music: a Mathematical Offering.*
December 2008, http://homepages.abdn.ac.uk/mth192/pages/html/maths-music.html

[3] E-mu Systems. *SoundFont Technical Specification.*
Version 2.01, July 1998 - http://connect.creativelabs.com/developer/SoundFont/sfspec21.pdf

[4] MIDI Manufacturers Association. *Downloadable Sounds Level 1.*
Version 1.1b, September 2004 - http://www.midi.org/techspecs/dls/dls1v11b.pdf

[5] MIDI Manufacturers Association. *General Midi System Level 1 Developer Guidelines – For Manufacturers and Composers.*
Second Revision, July 1998 - http://www.midi.org/techspecs/gmguide2.pdf

[6] Olli Niemitalo. *Polynomial Interpolators for High-Quality Resampling of Oversampled Audio.*
October 2001 - www.student.oulu.fi/~oniemita/dsp/deip.pdf

[7] Wikipedia
http://en.wikipedia.org/wiki/Main_Page

12. DOCUMENTS ANNEXES

ADMINISTRATIF

ADM0 *Planning & gestion du temps*

SCHEMATIQUE

SCH0 *Schémas logiques des unités du canal physique*
SCH1 *Schémas logiques des unités non implémentées du canal physique*
SCH2 *Schéma logique du « SPU Manager »*
SCH3 *Schéma logique du « Registers & Memory Access Manager »*
SCH4 *Schémas logiques des blocs relatifs aux accès mémoire*

13. CD-ROM

Le CD-ROM fourni avec ce rapport contient l'ensemble des documents imprimés, les codes sources, de la documentation et les résultats de ce projet. Il est structuré comme suit :

▶ 00_REPORT

Version numérique du rapport.

▼ 01_MAINWORK

- **▶ HSPU_WORKSPACE** Contient le *workspace* Quartus*II* et Eclipse du projet. Veuillez lire le fichier `README.TXT` avant toute utilisation.
- **▶ CODE_EXTRACT** Contient uniquement les codes sources VHDL et C++, extraits du workspace pour un accès rapide.

▼ 02_MISCWORK

- **▶ RAWTOTABLE** Petit utilitaire réalisé pour convertir n'importe quel fichier en tableau en vue d'une inclusion dans un code source C/C++.
- **▶ RESPCONV** Contient les fichiers nécessaires en vue du futur développement de l'unité « Response Converter » du canal physique.
- **▶ SQUAREHARP** Tous les fichiers de développement du synthétiseur «SquareHarp», à savoir : l'instrument ZynAddSubFX, son enregistrement et les conversions de celui-ci.

▼ 03_DOCUMENTATION

- **▶ ALTERA_DOCS** Documents de développement d'Altera (IP, Nios*II*, Avalon bus).
- **▶ BOOK_SPECS** Livres & spécifications.
- **▶ DE270_BOARD** Extraits du CD-ROM livré avec la carte de développement DE2-70. Contient un manuel, la schématique et tous les datasheets.

▼ 04_DEMOS

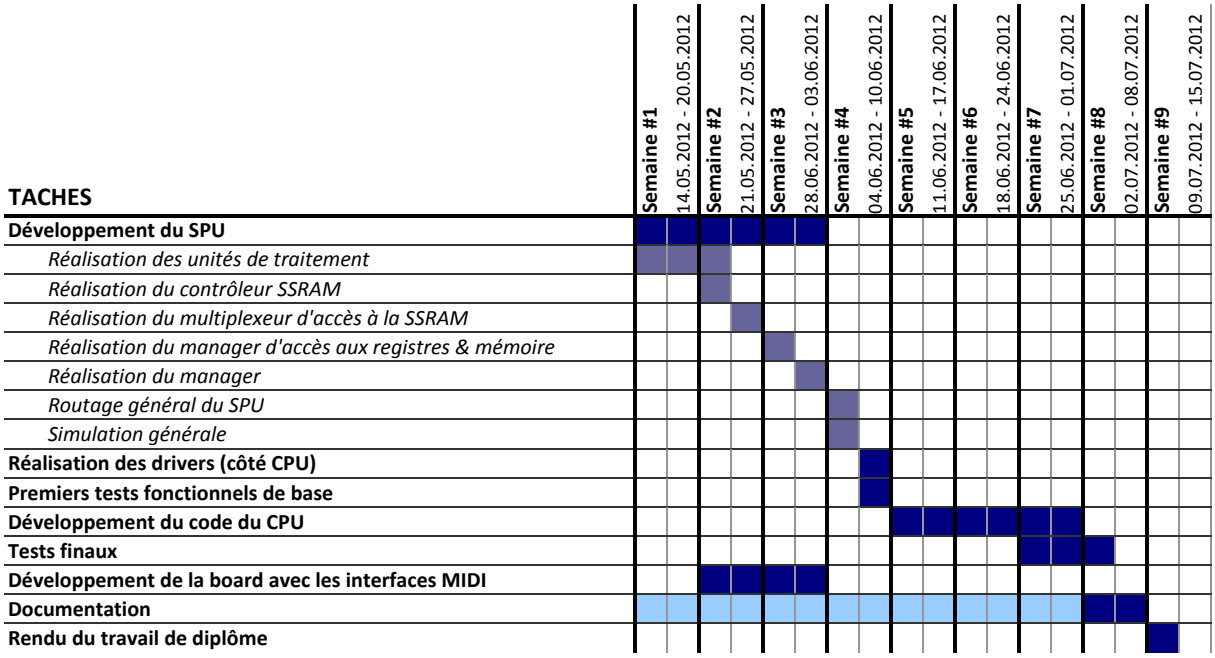
Enregistrements de démonstration du HSPU. Veuillez lire le fichier `DEMOLIST.PDF` pour une description détaillée de chaque enregistrement.

ADMINISTRATIF
Planning & Gestion du temps

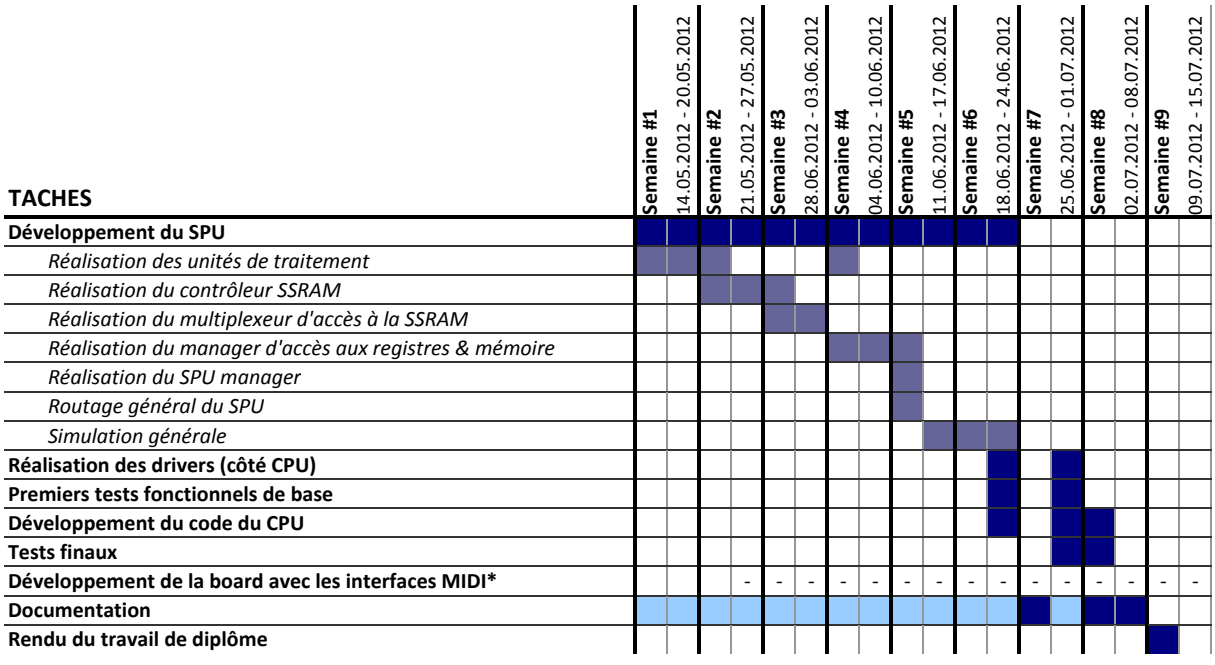
ADM0

PLANNING & GESTION DU TEMPS

PLANNING (établi le 21 mai 2012)



GESTION DU TEMPS



** Annulé en cours de projet.*

COMMENTAIRES

D'emblée, l'implémentation des fonctionnalités ADSR, LFO et d'un filtre (tous mentionnés dans la donnée du travail) a été jugée très difficile dans le temps imparti. Celles-ci ont donc été écartées.

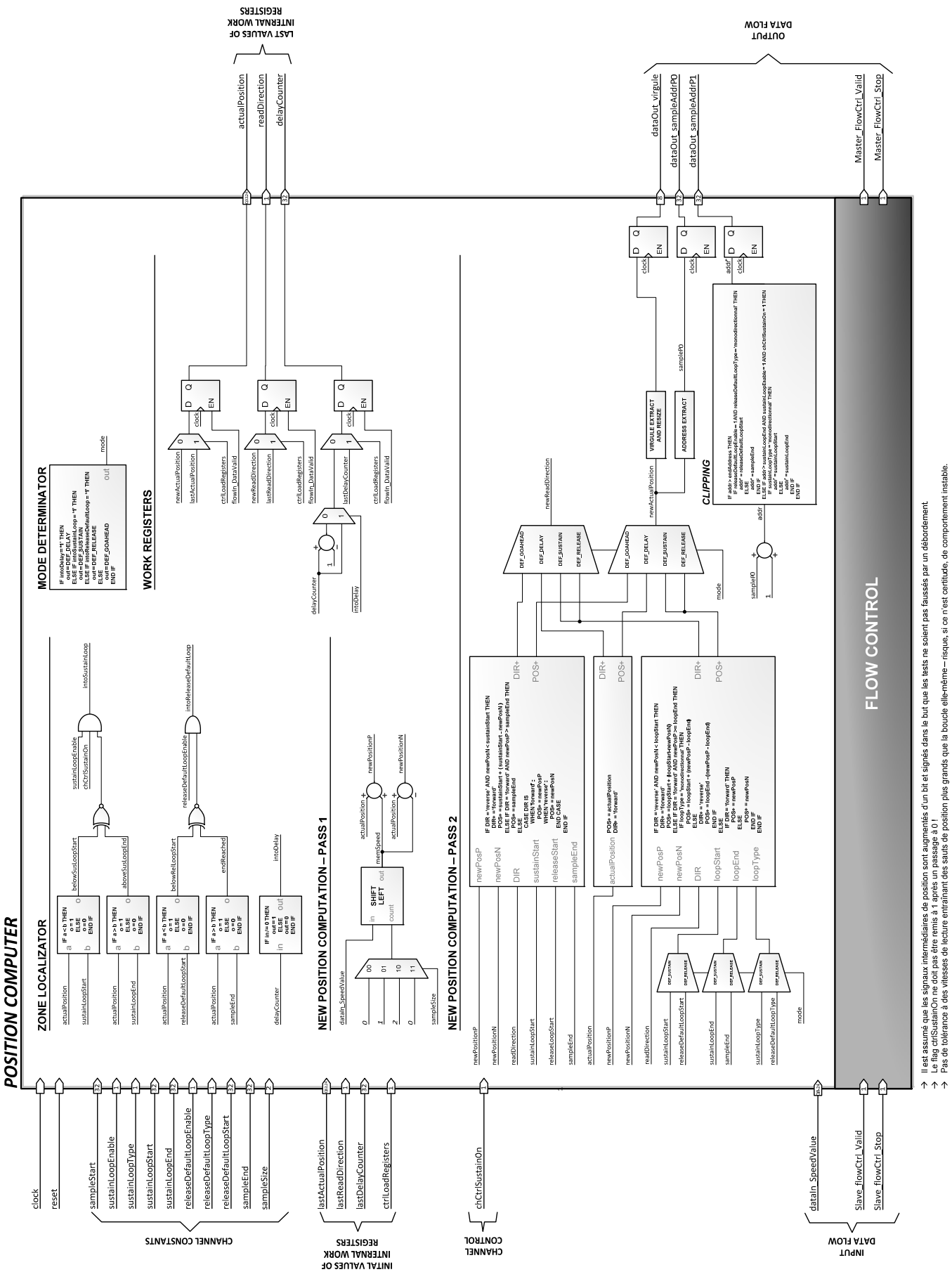
Le temps pour la réalisation du contrôleur SSRAM et du multiplexeur d'accès à la mémoire a été le double de ce qui était prévu. Celui de simulation (individuel et général) a également été légèrement sous-estimé. Cela a eu pour conséquence de réduire drastiquement le temps de développement du code du CPU (de 3 semaines à 1) : c'est ainsi que le démonstrateur final se limite à un synthétiseur MIDI mono-instrumental. Malgré cela, le projet a pu être mené à bien dans le temps imparti.

SCHEMATIQUE

Schémas logiques des unités du canal physique

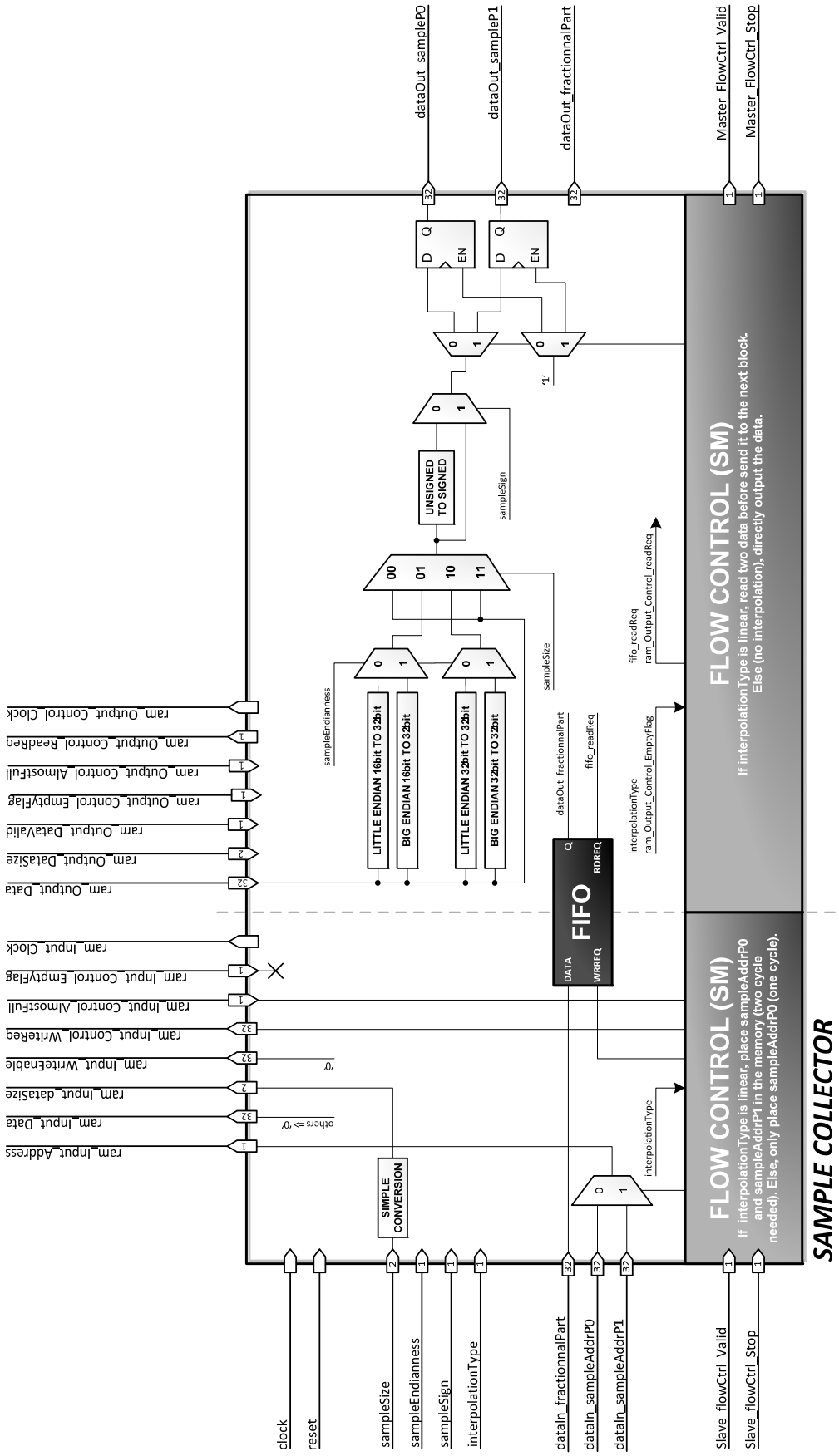
SCH0

POSITION COMPUTER

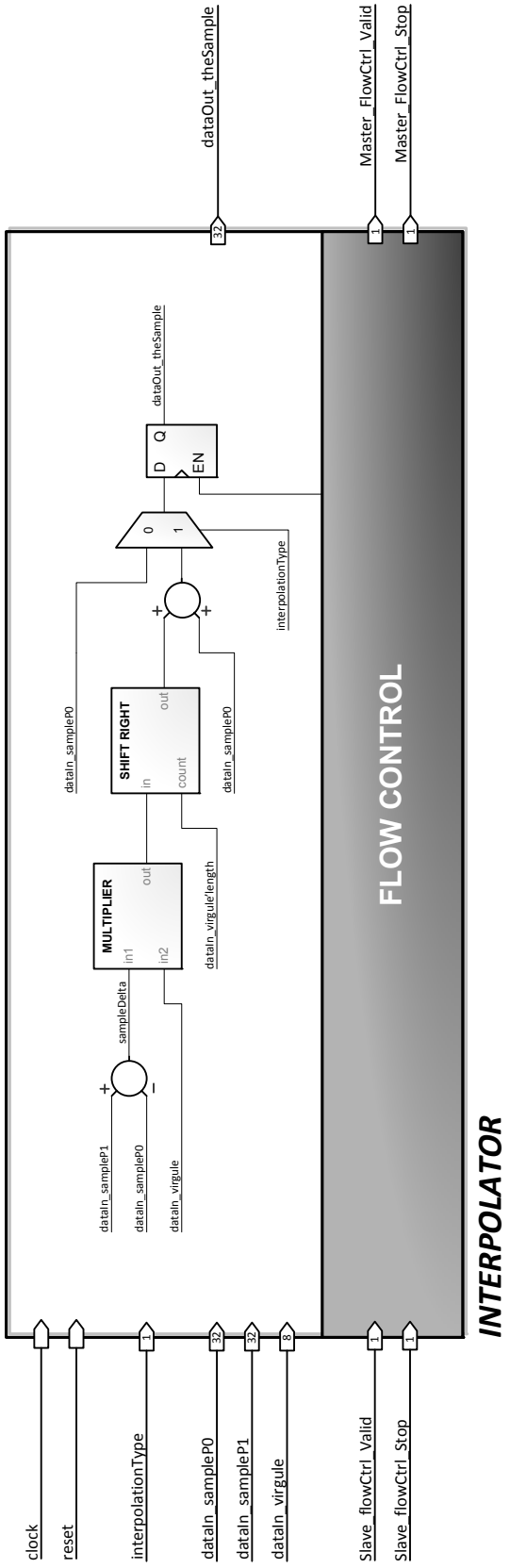


→ Il est assumé que les signaux intermédiaires de position sont augmentés d'un bit et signés dans le but que les tests ne soient pas faussés par un débordement.
→ Le flag ctrlSustainOn ne doit pas être remis à 1 après un passage à 0 !
→ Pas de tolérance à des vitesses de lecture entraînant des sauts de position plus grands que la boucle elle-même – risque, si ce n'est certitude, de comportement instable.

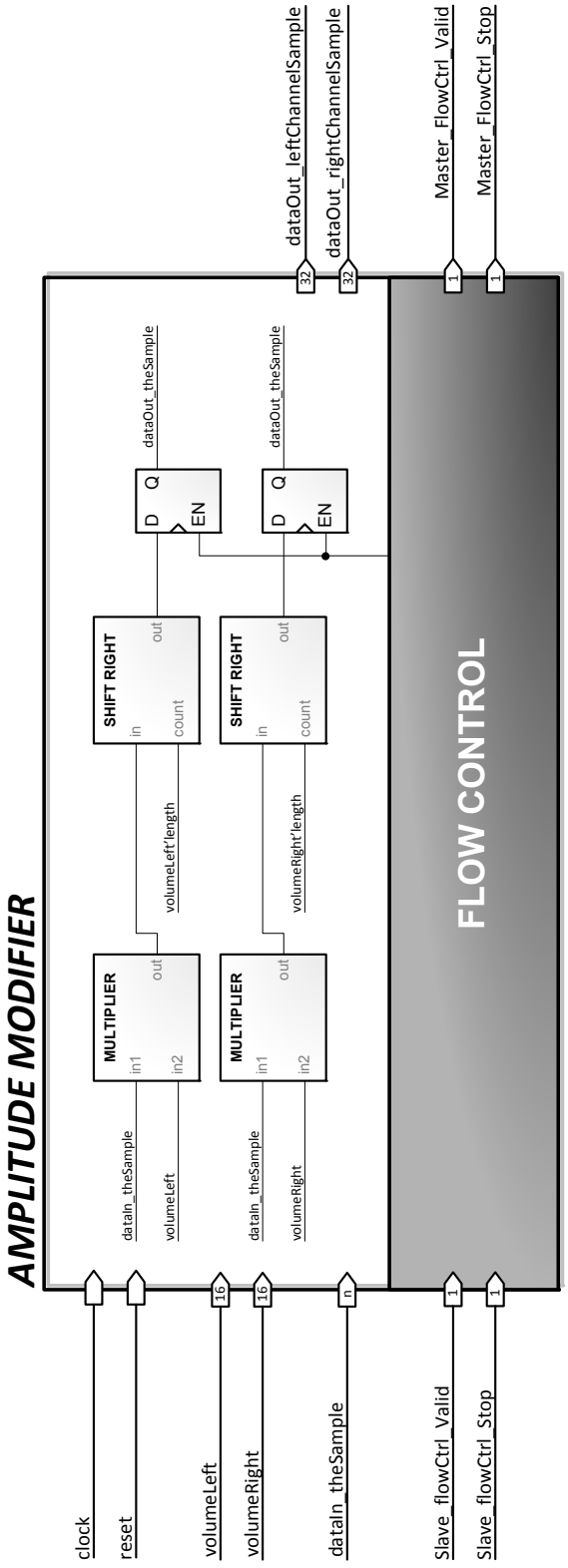
SAMPLE COLLECTOR



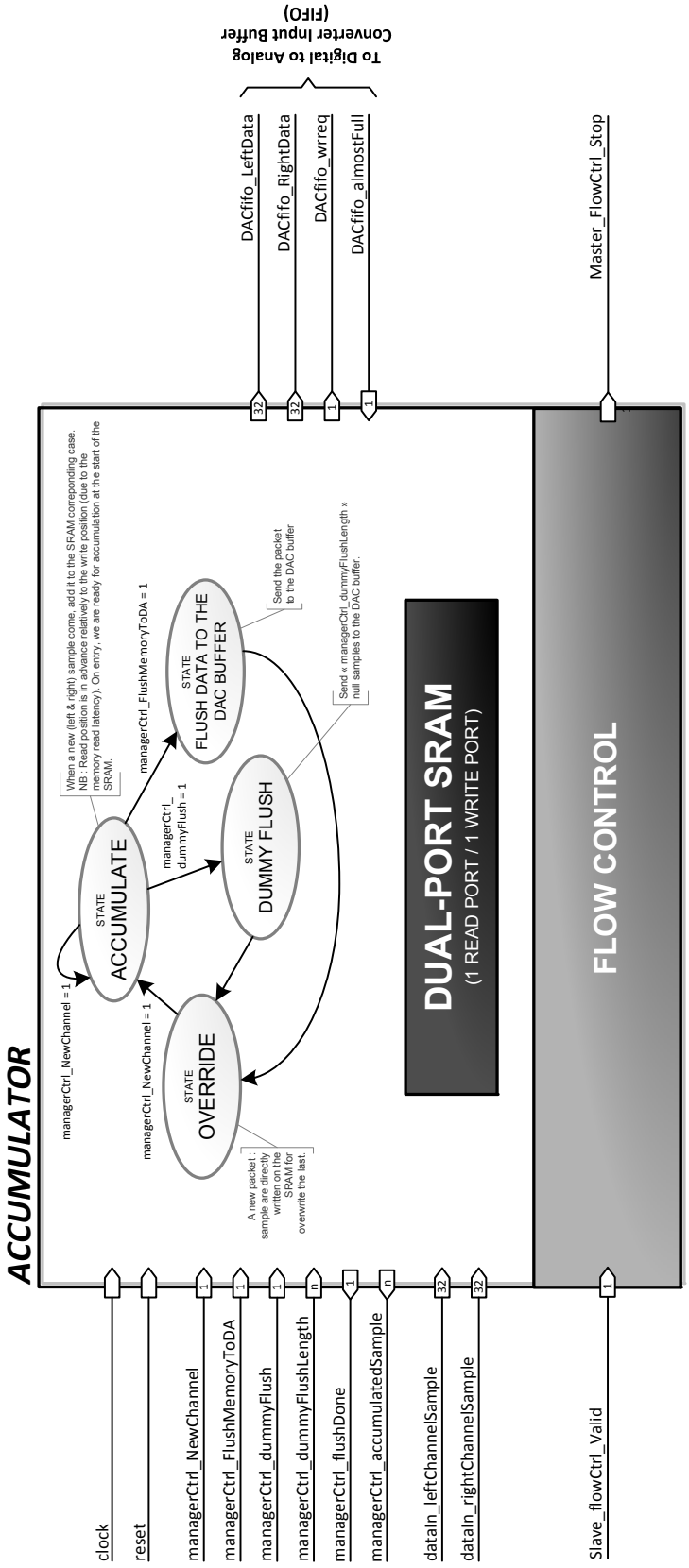
INTERPOLATOR



AMPLITUDE MODIFIER



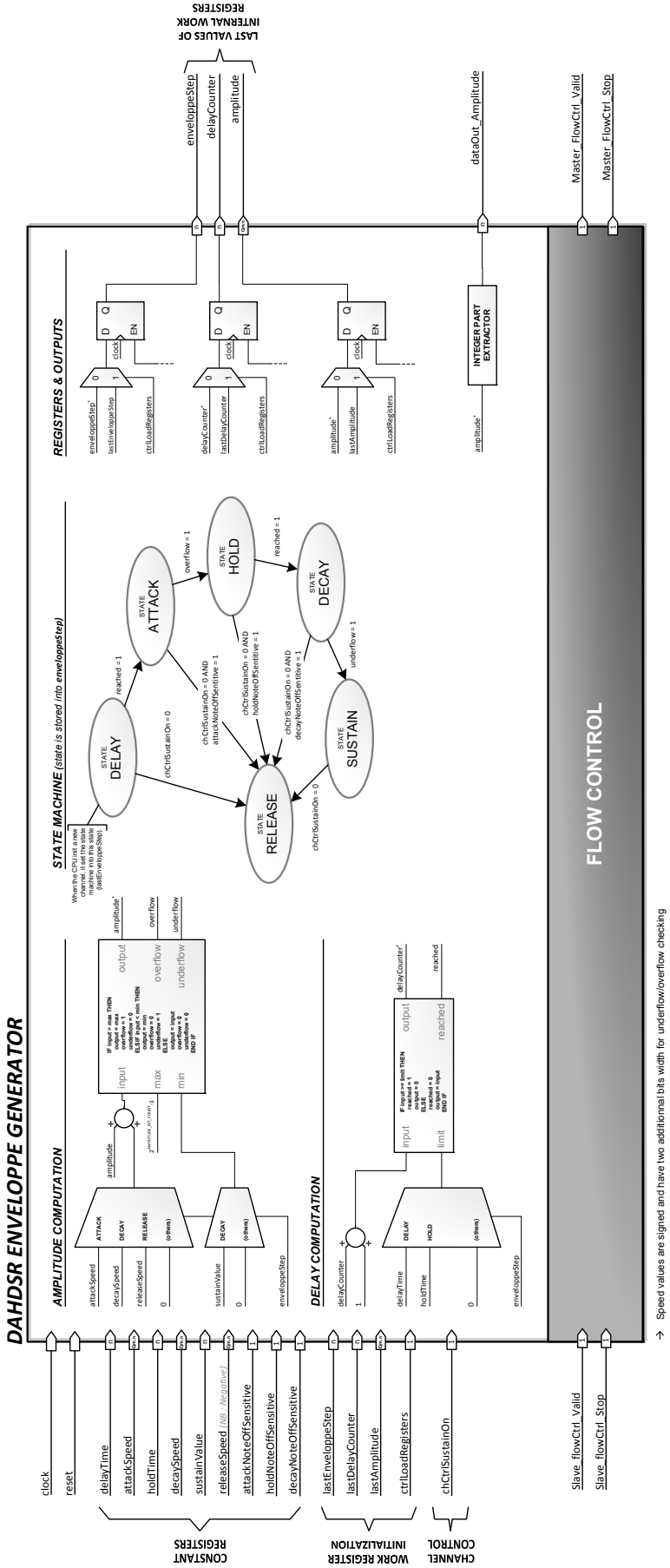
ACCUMULATOR



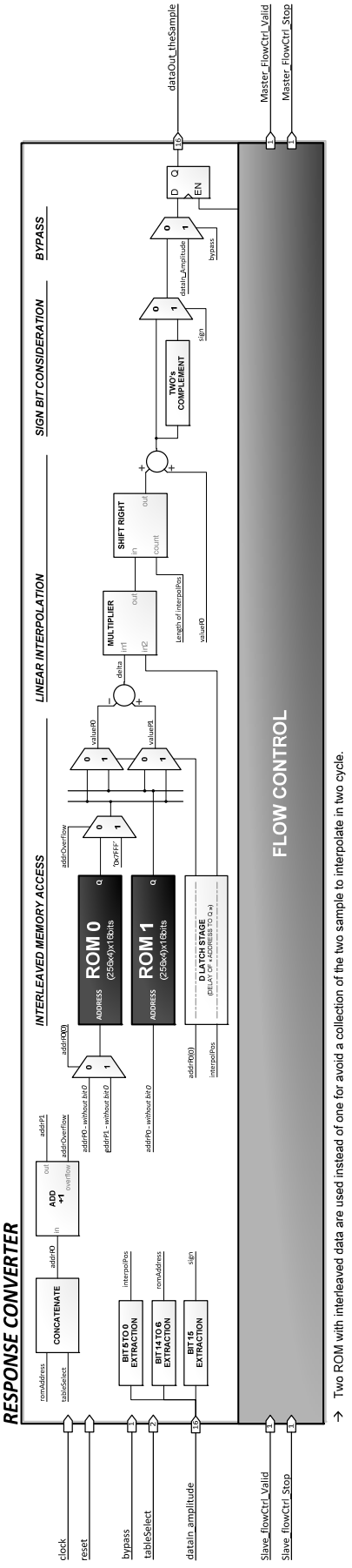
SCHEMATIQUE | SCH1

Schémas logiques des unités non implémentées du canal physique

DAHDSR ENVELOPPE GENERATOR

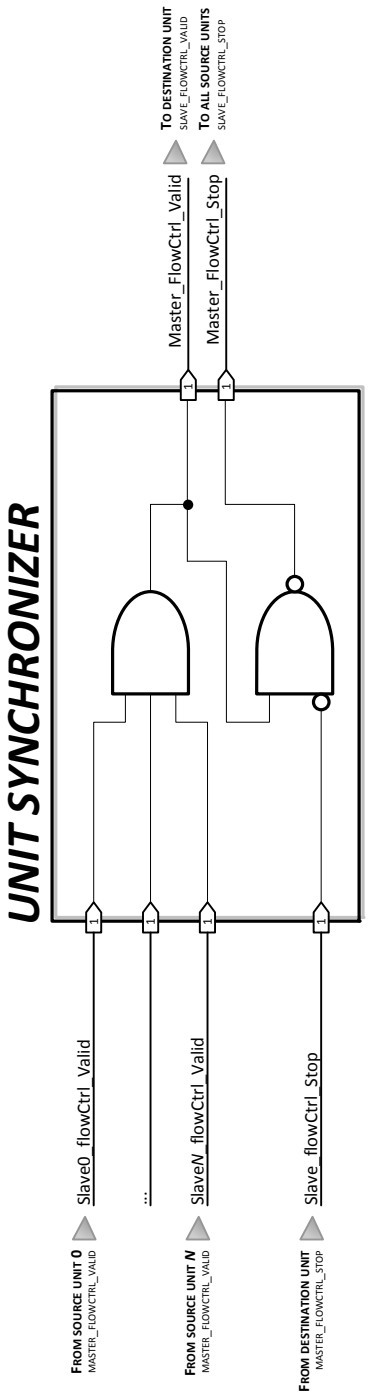


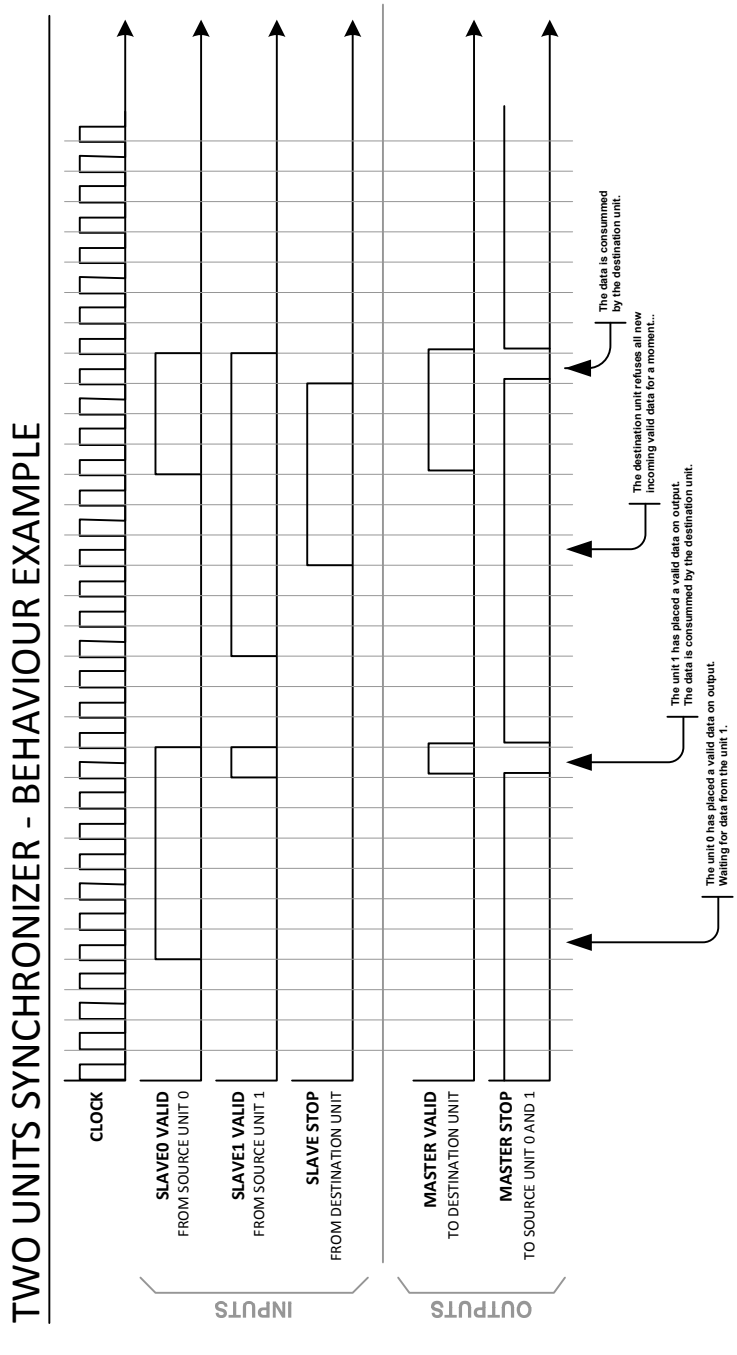
RESPONSE CONVERTER



→ Two ROM with interleaved data are used instead of one for avoid a collection of the two sample to interpolate in two cycle.

UNIT SYNCHRONIZER



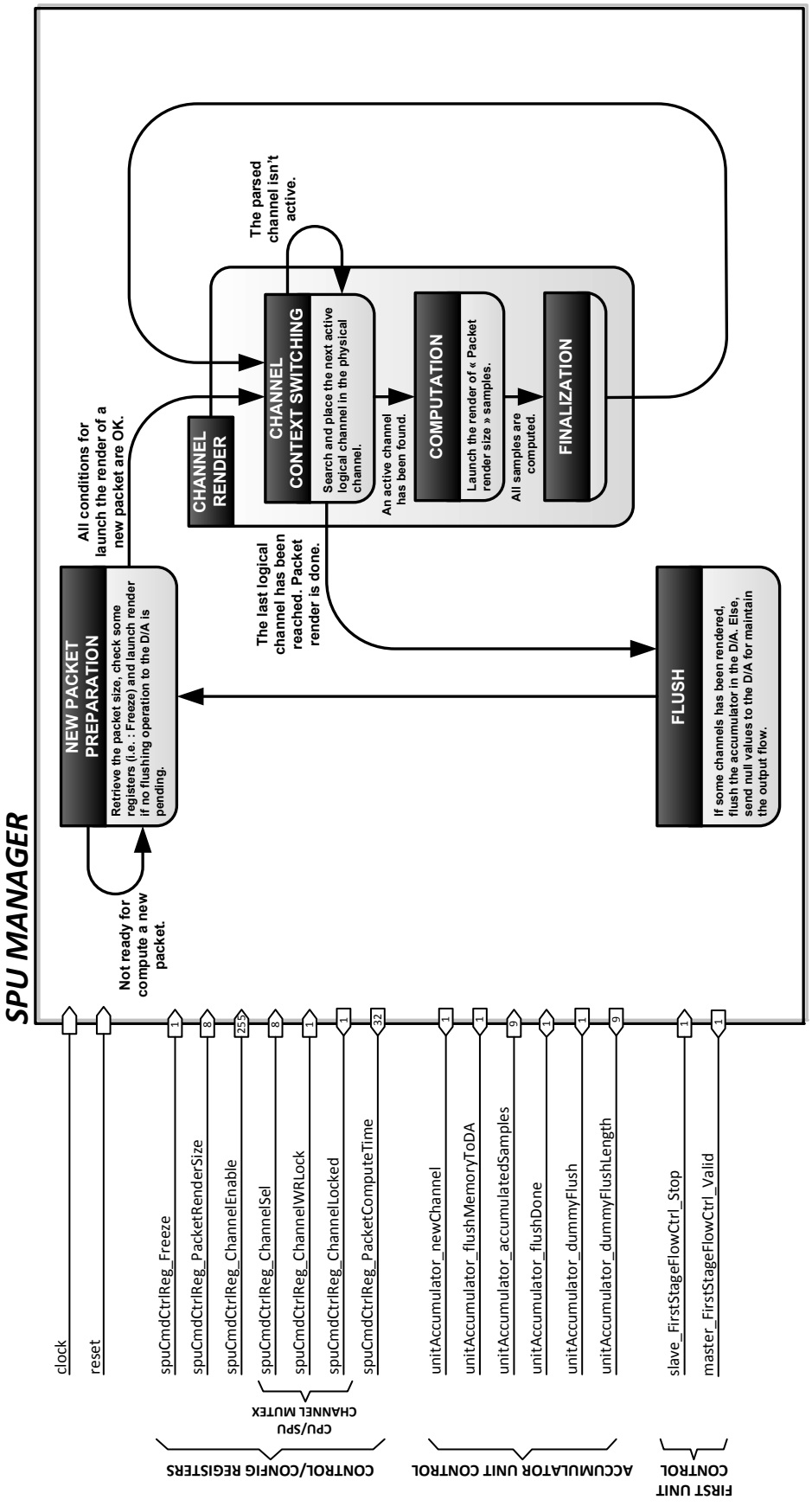


SCHEMATIQUE

Schéma logique du « SPU Manager »

SCH2

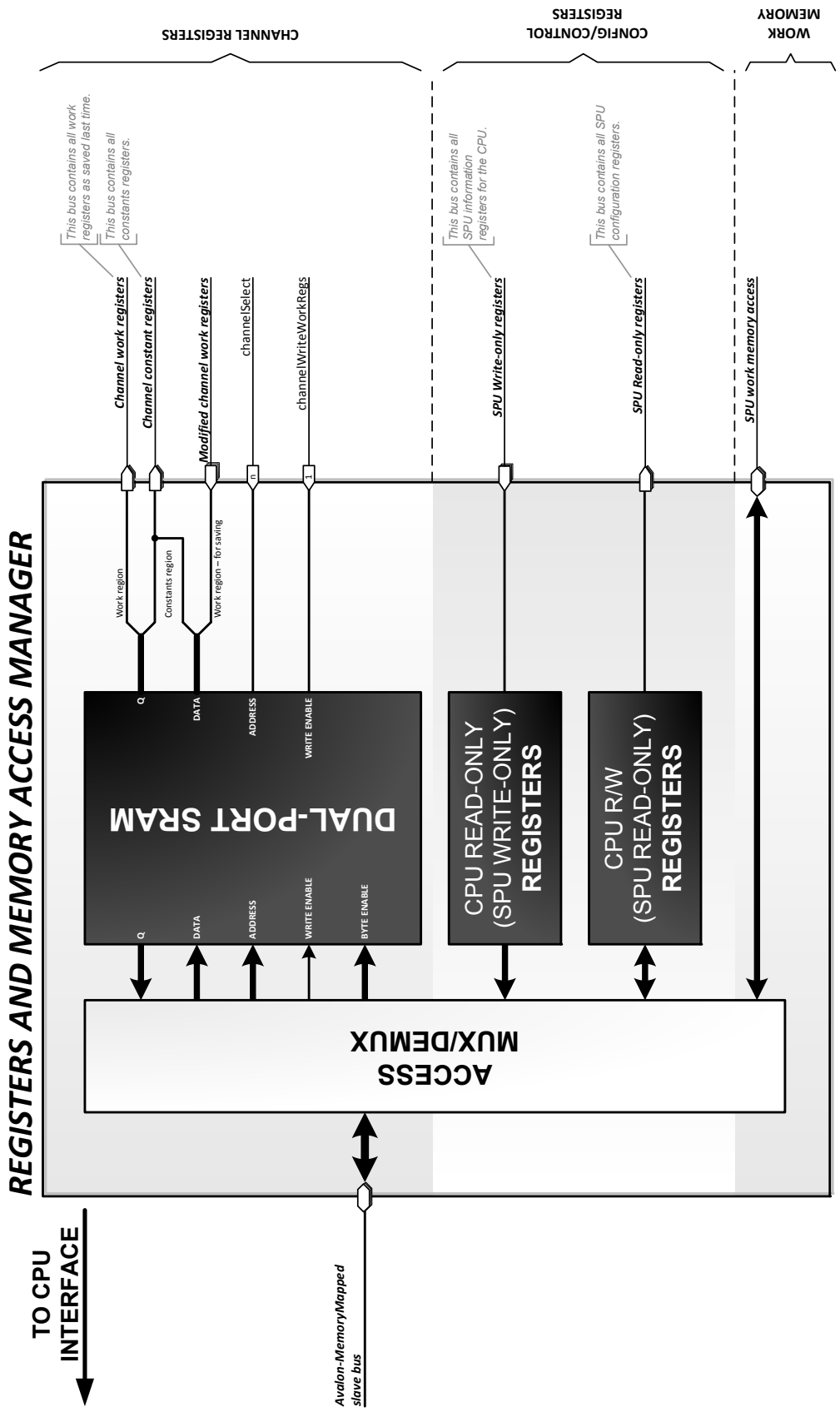
SPU MANAGER


SPU MANAGER
NEW PACKET PREPARATION
Retrieve the packet size, check some registers (i.e. : Freeze) and launch render if no flushing operation to the D/A is pending.
Not ready for compute a new packet.
All conditions for launch the render of a new packet are OK.
CHANNEL RENDER
CHANNEL CONTEXT SWITCHING
Search and place the next active logical channel in the physical channel.
The parsed channel isn't active.
An active channel has been found.
COMPUTATION
Launch the render of « Packet render size » samples.
All samples are computed.
FINALIZATION
The last logical channel has been reached. Packet render is done.
FLUSH
If some channels has been rendered, flush the accumulator in the D/A. Else, send null values to the D/A for maintain the output flow.
clock
reset
spuCmdCtrlReg_Freeze
spuCmdCtrlReg_PacketRenderSize
spuCmdCtrlReg_ChannelEnable
spuCmdCtrlReg_ChannelSel
spuCmdCtrlReg_ChannelWRLock
spuCmdCtrlReg_ChannelLocked
spuCmdCtrlReg_PacketComputeTime
unitAccumulator_newChannel
unitAccumulator_flushMemoryToDA
unitAccumulator_accumulatedSamples
unitAccumulator_flushDone
unitAccumulator_dummyFlush
unitAccumulator_dummyFlushLength
slave_FirstStageFlowCtrl_Stop
master_FirstStageFlowCtrl_Valid
CPU/SPU CHANNEL MUTEX
CONTROL/CONFIG REGISTERS
ACCUMULATOR UNIT CONTROL
FIRST UNIT CONTROL

SCHEMATIQUE | SCH3

Schéma logique du « Registers & Memory Access Manager »

REGISTERS AND MEMORY ACCESS MANAGER



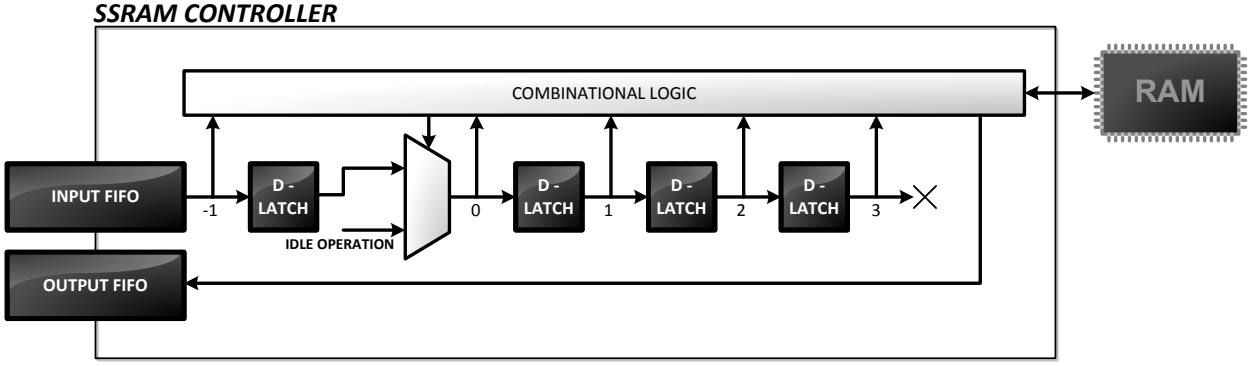
SCHEMATIQUE

Schémas logiques des blocs relatifs aux accès mémoire

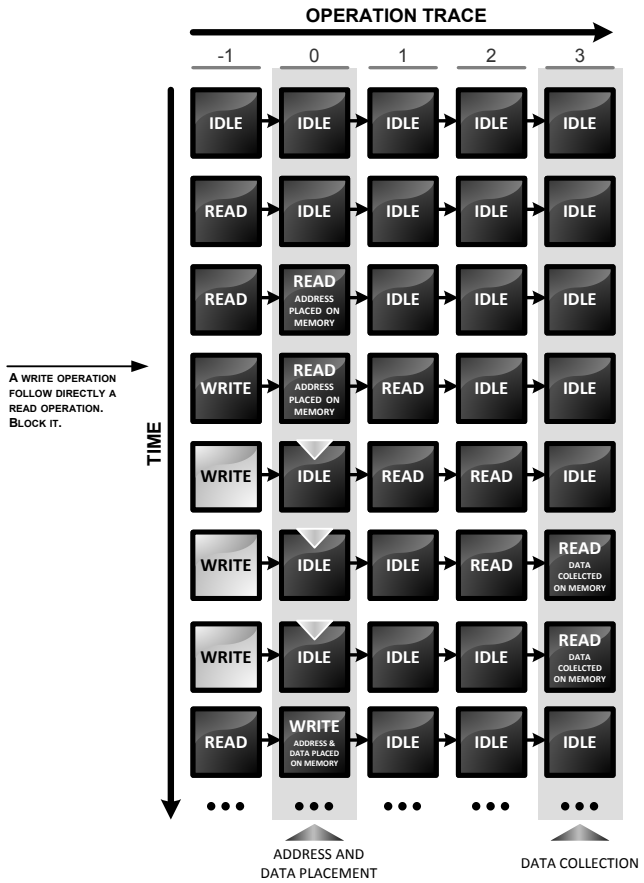
SCH4

SSRAM CONTROLLER

SSRAM CONTROLLER



BEHAVIOUR EXAMPLE



MEMORY ACCESS MULTIPLEXER